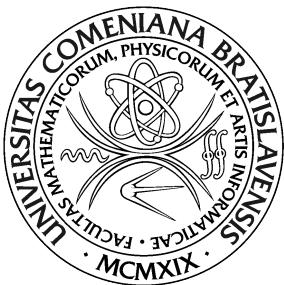


UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



VIZUÁLNY SYSTÉM PRE
INTERAKCIU ĽUDSKÉHO
UČITEĽA S HUMANOIDNÝM
ROBOTOM

Diplomová práca

2022

Bc. Nicolas Orság

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



VIZUÁLNY SYSTÉM PRE
INTERAKCIU ĽUDSKÉHO
UČITEĽA S HUMANOIDNÝM
ROBOTOM

Diplomová práca

- Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Ing. Viktor Kocur, PhD.

Bratislava, 2022

Bc. Nicolas Orság



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Nicolas Orság

Študijný program: aplikovaná informatika (Jednoodborové štúdium,
magisterský II. st., denná forma)

Študijný odbor: informatika

Typ záverečnej práce: diplomová

Jazyk záverečnej práce: slovenský

Sekundárny jazyk: anglický

Názov: Vizuálny systém pre interakciu ľudského učiteľa s humanoidným robotom
Visual system for interaction of a human teacher with a humanoid robot

Anotácia: Toto zadanie je súčasťou projektu interakcie ľudského učiteľa s robotom. Robot pri tejto interakcii manipuluje jednoduchými objektmi na základe pokynov od ľudského učiteľa. Pre tento účel je tak vhodné aby robot dokázal správne detegovať pozíciu jednoduchých, objektov, učiteľovú a svoje ruky. Následne je potrebné aby robot dokázal rozpoznať zadané inštrukcie.

Ciel: Cieľom tejto práce je navrhnúť, implementovať a otestovať systém ktorý na základe vstupných stereo dát z kamier v robotovi NICO deteguje pozíciu jednoduchých objektov, učiteľovej ruky a robotových rúk a následne rozpozná gestá od učiteľa. Súčasťou práce bude prehľad existujúcich riešení detekcie objektov v stereo snímkach a rozpoznávaní gest učiteľa. Navrhnutý systém bude vyhodnotený v kontexte prebiehajúceho projektu interakcie ľudského učiteľa s robotom.

Vedúci: Ing. Viktor Kocur, PhD.

Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 05.10.2021

Dátum schválenia: 06.10.2021

prof. RNDr. Roman Ďuríkovič, PhD.

garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

.....

Bratislava, 2022

Bc. Nicolas Orság

Pod'akovanie

Rád by som vyjadril úprimnú vdaku tým, ktorí mi pomohli pri dokončení tejto diplomovej práce.

Najprv by som chcel pod'akovať svojmu vedúcemu práce, Ing. Viktorovi Kocurovi, PhD., za jeho cenné rady, odborné vedenie a trpezlivosť. Vždy mi bol nápomocný a reagoval veľmi rýchlo, jeho spätnú väzbu so dostával vždy takmer okamžite.

Rovnako by som chcel pod'akovať svojim rodinným príslušníkom a priateľom za ich podporu a povzbudenie v priebehu celého procesu tvorby tejto práce. Ich podpora bola pre mňa veľmi dôležitá.

Nakoniec by som chcel pod'akovať všetkým ľuďom, ktorí mi pomohli priamo alebo nepriamo s touto prácou.

Ešte raz by som sa chcel pod'akovať všetkým, ktorí mi pomohli s touto diplomovou prácou. Bez Vášho prínosu by táto práca nebola možná.

Abstrakt

Oblast' interakcie človeka a robota výrazne pokročila v posledných rokoch, pričom humanoidné roboty sa vyvíjajú pre široké spektrum aplikácií. Efektívna interakcia medzi týmito robotmi a ich používateľmi je však stále výzvou. Táto práca sa zameriava na few-shot algoritmy pre objektovú detekciu, ktoré používajú iba malé množstvo anotovaných dát na tréning, čím minimálizujú náklady a čas potrebné na anotáciu. Skúmame algoritmus Frustratingly Simple Few-Shot Object Detection [1], testujeme jeho presnosť a rýchlosť pri rôznom počte trénovacích obrázkov na datasete PASCAL VOC [2] a pri rôznom počte nových tried, ktoré sme získali z roboflow [3]. Výsledkom našej práce je 8-násobné zrýchlenie a výrazné zníženie pamäťovej náročnosti algoritmu na GPU, pri zachovaní rovnakej presnosti. Nakoniec sme náš zrýchlený algoritmus otestovali na snímkoch z kamery robota NICO, kde sme dosiahli presnosť AP₅₀ = 18.182.

Kľúčové slová: Počítačové videnie, Humanoidný robot, Hlboké učenie, Konvolučné neurónové siete, Few-shot object detection

Abstract

The field of human-robot interaction has made significant progress in recent years, with humanoid robots being developed for a wide range of applications. However, effective interaction between these robots and their users remains a challenge. This work focuses on few-shot algorithms for object detection, which use only a small amount of annotated data for training, thus minimizing the costs and time required for annotation. We explore the Frustratingly Simple Few-Shot Object Detection [1] algorithm, test its accuracy and speed with varying numbers of training images on the PASCAL VOC [2] dataset and with varying numbers of new classes obtained from roboflow [3]. The result of our work is an 8-fold speedup and significant reduction in the algorithm's memory requirements on the GPU, while maintaining the same accuracy. Finally, we tested our accelerated algorithm on images from the NICO robot's camera, achieving an mAP50 accuracy of 18.182.

Keywords: Computer vision, Humanoid robot, Deep learning, Convolutional neural networks, Few-shot object detection

Obsah

1	Úvod do problematiky	3
1.1	Počítačové videnie	3
1.2	Príznaky	4
1.3	Objektová detekcia	5
1.3.1	Tradičné metódy	5
1.3.2	Metódy hlbokého učenia	6
1.4	Few-shot object detection(FSOD)	19
1.4.1	Prístupy k FSOD	20
1.4.2	Datasets pre vyhodnotenie FSOD	21
1.4.3	Aktuálne riešenia FSOD	22
2	Testovanie FSFSODT	24
2.1	Dataset	24
2.2	Trénинг	26
2.3	Testovanie rýchlosťi a presnosti modelu pri zmene počtu trénovacích obrázkov novel tried	27
2.3.1	1-shot detekcia	27
2.3.2	2-shot detekcia	29
2.3.3	3-shot detekcia	31
2.3.4	5-shot detekcia	32

<i>OBSAH</i>	ix
--------------	----

2.3.5 10-shot detekcia	34
2.3.6 15-shot detekcia	35
2.3.7 Vyhodnotenie	37
3 Zrýchlenie FSFSOD	40
3.1 Zrýchlenie pomocou zmeny parametrov	40
3.2 Zrýchlenie pomocou zapamätania si výstupu zo zmrazených vrstiev	43
3.2.1 Zapamätanie si výstupu z backbone	44
3.2.2 Zapamätanie si výstupu priamo z Box Head	50
4 Testovanie algoritmu pri zmene počtu tried	55
4.1 Fine-tuning na base + novel triedach	55
4.2 Fine-tuning len na novel triedach	70
5 Testovanie tréningu modelu pre detekciu objektov pomocou kamery robota NICO	72
5.1 Testovanie na pôvodných snímkoch z kamery	72
5.2 Testovanie na snímkoch z kamery zbavených skreslenia	77

Úvod

Detekcia objektov je kľúčovým úlohou v oblasti počítačového videnia s rôznymi aplikáciami vo vybraných oblastiach, ako sú robotika, dozor a autonomné riadenie. Tradičné algoritmy detekcie objektov spočívajú v potrebe veľkého množstva anotovaných trénovacích vzoriek na dosiahnutie vysokého stupňa presnosti. Získanie takýchto datasetov môže byť náročné a nákladné, najmä v prípadoch, kde sa objekty záujmu vyskytujú zriedkavo alebo sú jedinečné. V posledných rokoch sa ako sľubný prístup na riešenie tohto problému ukázalo učenie s malým množstvom vzoriek (few-shot learning).

Few-shot learning sa snaží umožniť učenie nových konceptov s obmedzeným množstvom anotovaných dát. V tejto práci sa zameriavame na few-shot detekciu objektov, ktorá zahrňa detekciu objektov záujmu pomocou len niekoľkých anotovaných obrázkoch. Naším cieľom je preskúmať few-shot algoritmy objektovej detekcie a vybrať taký, ktorý bude možno trénovať efektívne. Následne otestujeme jeho rýchlosť a presnosť pre rôzny počet trénovacích obrázkov a rôzny počet detegovaných tried. Pokúsime sa ho zefektívniť pre naše potreby, natrénovať a otestovať model pre few-shot objektovú detekciu na záberoch z kamery humanoidného robota NICO.

V kapitole 1 si vysvetlíme základné pojmy a princípy počítačového videnia a objektovej detekcie. V kapitole 2 otestujeme presnosť a rýchlosť algoritmu Frustratingly Simple Few-Shot Object Detection [1] po pridaní novej triedy

pri rôznom počte trénovacích obrázkov. V kapitole 3 spravíme niekoľko pokusov s cieľom zrýchliť tento algoritmus, čo sa nám nakoniec úspešne podarilo. V kapitole 4 otestujeme náš zrýchlený algoritmus pri rôznom počte nových tried. V poslednej kapitole 5 otestujeme náš zrýchlený algoritmus na vlastnom datasete, ktorý tvoria snímky z kamery robota NICO.

Kapitola 1

Úvod do problematiky

V úvodnej kapitole sa budeme venovať základným pojmom, ktoré sú nevyhnutné pre pochopenie nášho výskumu. Vysvetlíme si ako funguje počítačové videnie, objektová detekcia, konvolučné neuronové siete a taktiež sa zameziame na výskum objektovej detekcie pri malej trénovacej množine (Few shot object detection), ktorý budeme chcieť využiť v našej práci pre učenie nových objektov aj z veľmi malého množstva dát.

1.1 Počítačové videnie

V súčasnosti predstavuje počítačové videnie v informatike veľmi rýchlo rastúci a progresívny smer. Snaží sa priblížiť vnímaniu sveta z pohľadu ľudského oka, ktoré je pre nás prirodzené a automaticky sme schopný rozpoznávať objekty, farby a kontext toho čo vidíme. Avšak plné sémanticke pochopenie videnej reality je veľmi komplexné a zatiaľ nie sme schopný ho získať spracovaním digitálneho obrazu. Hlavne preto, že pochopenie obrazu môže vyplývať zo súvislostí, ktoré nie sú súčasťou obrazu.

Avšak počítačové videnie sa posúva veľmi rýchlo vpred. Neustále vyni-

kajú nové algoritmy a prístupy či už na detekciu objektov alebo klasifikáciu obrazu. Medzi základné problémy počítačového videnia patrí klasifikácia, objektová detekcia a segmentácia. Pri klasifikácii sa snažíme obraz priradiť do jednej z tried. V objektovej detekcii sa snažíme v obraze určiť oblasti všetkých známych objektov a priradiť ich do tried. A pri segmentácii je našim cieľom rozdeliť obraz do viacero oblastí a každému pixlu určiť oblasť do ktorej patrí.

1.2 Príznaky

Pri riešení problémov v počítačovom videní sa využívajú príznaky. Príznak v počítačovom videní je merateľný kus dát v obrázku, ktorý je unikátny pre špecifický objekt. Príznak môže reprezentovať napríklad štýl sfarbenia, nejaký tvar, či už čiaru alebo hranu v obraze alebo nejakú časť obrazu. Vďaka dobrému príznaku dokážeme od seba rozlísiť objekty. Napríklad ak máme rozlísiť mačku a bicykel tak ako dobrý príznak by mohlo byť, že na obrázku sa nachádza koleso. Hneď by sme vedeli vďaka tomuto príznaku klasifikovať obrázok do týchto dvoch tried. Ak by sme však mali za úlohu zistiť či je na obrázku motorka alebo bicykel, tak by nám tento príznak veľmi nepomohol a museli by sme pozerať na iné príznaky. Preto zväčša neextrahujeme z obrázku len jeden príznak, ale pre lepsiu detekciu vyberáme viacej príznakov, ktoré tvoria príznakový vektor.

Nie je presná definícia aké príznaky obrázku by sme mali použiť, ale závisí to skôr od nášho cieľu a typu úlohy. Príznaky sa delia na lokálne a globálne. Príznaky, ktoré popisujú celý obrázok, sa nazývajú globálne príznaky. Napríklad ako ako veľmi sú dominantné jednotlivé farby v obrázku. Globálny príznak nám opisuje obraz ako celok a mal by reporezentovať nejakú jeho špecifickú vlastnosť. Lokálne príznaky sa extrahujú len z určitej zaujímavej ob-

lasti v obrázku, využívajú sa najmä pri objektovej detekcii. Najskôr nájdeme zaujímavé oblasti, ktoré by mohli reprezentovať nejakú zaujímavú vlastnosť alebo nejaký objekt. Následne vytvoríme príznakový vektor pre danú oblasť, ktorý by nám mal poskytnúť zásadnú informáciu o tejto časti obrazu. Treba rátať s tým, že objekt na obrázku môže byť rôznej veľkosti, rôzne natočený, rôzne osvetlený, zašumený, môže sa nachádzať v rôznych častiach obrázku a podobne. Preto naše príznaky by mali byť ideálne invariantné voči týmto zmenám.

1.3 Objektová detekcia

Jedným z najskúmanejších problémov v počítačovom videní je objektová detekcia, ktorá spočíva v rozpoznaní jednotlivých objektov a ich pozícii v digitálnom obraze. K tomuto problému sa dá pristupovať tradičnými metódami počítačového videnia, alebo dnes už veľmi rozšíreným s oveľa lepšími a presnejšími výsledkami, ako pri tradičných metódach a to pomocou hlbokého učenia, ktorých klúčom je naučiť sa na veľkých dátach extrahovať príznaky, tak aby mala detekcia čo najväčšiu presnosť.

1.3.1 Tradičné metódy

Ako prvé vznikli tradičné metódy. Vysvetlíme si ako fungujú, pretože nám to pomôže pochopiť ako funguje dnes najvýužívanejší a najpresnejší prístup hlbokého učenia na ktorý sa zameriame v tejto práci. Tradičné metódy v objektovej detekcii majú zvyčajne tri etapy: vybranie oblasti, extrakcia príznakov, klasifikácia objektu.

V prvej etape sa snažíme lokalizovať objekt. Keďže objekt môže byť rôznej veľkosti, musíme skenovať celý obrázok pomocou posúvneho okna rôznej

veľkosti. Táto metóda je výpočtovo náročná.

V druhej etape použijeme použijeme metódy ako SIFT [4], HOG [5] na extrakciu vizuálnych príznakov na rozpoznanie objektu. Tieto príznaky nám poskytujú sémantickú a robustnú reprezentáciu. Avšak kvôli rôznemu osvetleniu, pozadiu a ulhu pohľadu je veľmi náročné manuálne navrhnuť deskriptor príznakov, ktorý by dokonale opísal všetky typy objektov.

V tretej fáze klasifikácie objektu používame zväčša Support Vector Machine(SVM) [6] alebo Adaboost [7] pre klasifikáciu cieľových objektov zo všetkých kategórii aby bola reprezentácia viac hierarchická, sémantická a informatívnejšia pre vizuálne rozpoznávanie.

Tradičné metódy majú problém s vysokou výpočtovou náročnosťou pri generovaní kandidátov na bounding box (obdlžník ohraničujúci objekt) pomocou posuvného okna a manuálnym nastavením extrakcie príznakov, ktoré nie je vždy veľmi presné. Napriek tomu majú výhodu, že nepotrebuju veľký anotovaný dataset ani vysokú výpočtovú silu pri tréningu, ktorú vyžaduje prístup hlbokého učenia.

1.3.2 Metódy hlbokého učenia

Neskôr keď tradičné metódy začali stagnovať, sa začali na riešenie problémov klasifikácie obrázkov, objektovej detekcie a segmentácie využívať metódy hlbokého učenia. Hlavným dôvodom, prečo metódy hlbokého učenia dosahujú lepšie výsledky ako tradičné metódy je, že netreba manuálne voliť príznaky, ale ich úlohou je nájsť najlepšie príznaky pre danú úlohu. Využívajú na to neurónové siete.

Neurónové siete

Základným stavebným blokom neurónovej siete je neurón. Vysvetlíme si ako funguje neurónova sieť zložená len z jedného neurónu. Do neurónu vstupuje m vstupov ktoré predstavuje vstupný vektor $\vec{x} = (x_1, x_2, \dots, x_m)$. Každý z týchto vstupov má svoju váhu. Váhy reprezentujeme váhovým vektorom $\vec{w} = (w_1, w_2, \dots, w_m)$. Najprv spravíme skalárny súčin vstupného a váhového vektora a prirátame k tomu bias b . Následne aplikujeme do nelienárnej aktivačnej funkcie f a dostaneme výstup z neurónu. Hlavnou úlohou aktivačnej funkcie je zavedenie nelinearity.

$$y = f(\vec{w} \cdot \vec{x} + b) \quad (1.1)$$

Neurónová sieť sa skladá zväčša z viacej neurónov a viacerých vrstiev. Výstup neurónu z nižšej vrstvy môže byť vstupom do neurónu vo vyššej vrstve a ako sme si popísali vyššie má svoju váhu, ktorá popisuje silu spojenia medzi dvoma neurónmi.

Váhy \vec{w} a bias b sú parametre, ktoré sa pri tréningu neurónovej siete prispôsobujú, tak aby sa minimalizoval rozdiel medzi výstupom siete a očakávaným výstupom.

Najviac využívaným typom neurónových sietí v počítačovom videní sú konvolučné neuronové siete, ktoré sú ideálne na spracovanie dát v mriežkovitom tvare ako napríklad obrázkov. O nich si povieme viac v ďalšej časti.

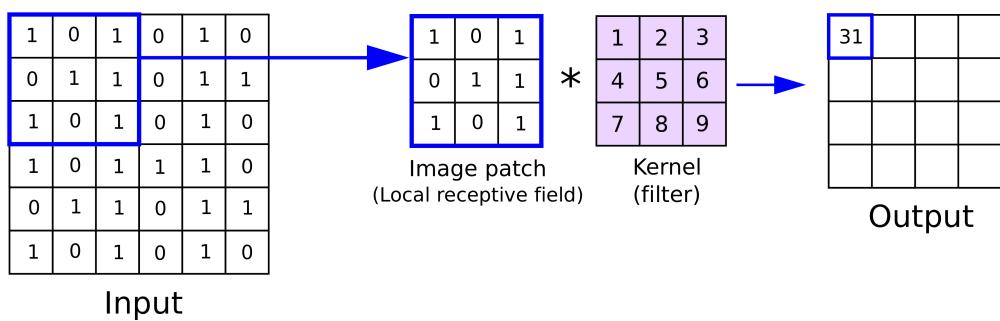
Úvod do CNN

Konvolučné neuronové siete (CNN) [8] sú typ neurónovej siete, ktorá obsahuje konvolučné vrstvy. Konvolučné vrstvy skenujú dátá pomocou množiny filtrov, kde každý filter hľadá špecifický vzor v dátach. Vďaka týmto vrstvám sú CNN

ideálne na spracovanie dát mriežkovitého tvaru ako napríklad obrázky. Teraz si vysvetlíme ako fungujú konvolučné vrstvy.

Konvolučné vrstvy

Ako sme si spomenuli v úvode, konvolúcia sa deje pomocou filtrov. Aplikáciu filtra si vysvetlíme pomocou konkrétneho príkladu na obrázku 1.1 kde máme vstupný obrázok veľkosti 6×6 a filter veľkosti 3×3 , po aplikácii tohto filtrov na obrázok dostaneme výstup veľkosti 4×4 . Na obrázku 1.1 vidíme vyrátanú prvú hodnotu na výstupe. Túto hodnotu dostaneme prenásobením hodnôt ľavého horného rohu veľkosti filtrov z obrazu (modré okno na obrázku) s hodnotami filtrov a následným sčítaním násobkov. Pre vyrávanie druhej hodnoty v prvom riadku sa naše modré okno veľkosti 3×3 , ktoré predstavuje hodnoty brané z obrázka posunie o 1 doprava a aplikujeme rovnaký postup ako pri výpočte prvej výstupnej hodnoty s novými hodnotami z obrázka. Takýmto spôsobom vyrátame všetky hodnoty v prvom riadku a následne pre ďalší riadok posunieme naše okno, ktoré berie hodnoty z obrázka o 1 nadol. Toto posunutie sa nazýva stride a dá sa nastaviť aj na vyššie číslo ako 1.



Obr. 1.1: Konkrétny príklad aplikácie filtrov na vstup

Ďalší nastaviteľný parameter pri konvolúcii je padding kde zväčšíme veľ-

kosť nášho vstupu tak, že pridáme rám okolo nášho vstupu s nulovými hodnotami ako vidíme na obrázku 1.2 bielou farbou pridané hodnoty pri paddingu veľkosti 1. Veľkosť paddingu predstavuje šírku tohto rámu. Padding by nemal byť väčší ako výška alebo šírka filtra. Vďaka paddingu vieme napríklad dosiahnúť, že rozmery výstupu budú rovnaké ako rozmery vstupu, pri rôznych rozmeroch filtra.

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

Obr. 1.2: Znázornenie paddingu pri konvolúcii

Pri nastavovaní rozmerov filtra, stridu a paddingu, musí ich kombinácia sedieť na rozmery vstupu, tak aby sa dal celý vstup prejsť filtrom.

Vstupný obrázok môže mať viacero kanálov, zväčša má 3 kanály, každý pre jednu z farieb červená, modrá a zelená, teda tvorí ho tenzor tretieho rádu. Filter teda bude tiež tenzor tretieho rádu a jeho posledná dimenzia bude rovnako veľká ako posledná dimenzia vstupu. Jeden filter nám teda vráti tenzor tretieho rádu s veľkosťou poslednej dimenzie 1. Napríklad máme

vstupný obraz veľkosti $32 \times 32 \times 3$, padding = 0, stride = 0 a filter veľkosti $5 \times 5 \times 3$, po aplikácii filtra dostaneme výstup veľkosti $28 \times 28 \times 1$.

Filtrov môže byť pri konvolúcii kludne aj viac ako 1. Počet filtrov nám nastavuje veľkosť poslednej dimenzie výstupu. Napríklad pri vstupe $6 \times 6 \times 3$ a použití 5 tich filtrov veľkosti $3 \times 3 \times 3$ stride = 1 a padding = 1 dostaneme výstup s rozmermi $6 \times 6 \times 5$. Pri ďalšej konvolučnej vrstve bude musieť byť posledná dimenzia veľkosti filtrov = 5. V ďalšej časti si predstavíme pooling.

Pooling

Po konvolučnej vrstve sa v CNN zvyčajne nachádza poolingová vrstva, ktorá zvyčajne zmenšuje priestorovú dimenziu. Poznáme niekoľko typov poolingu, vrátane max poolingu a average poolingu, ale najbežnejším je max pooling, ktorý vezme maximálnu hodnotu každého poolingového regiónu.

Napríklad, ak pooling región je mriežka 2×2 , max pooling vezme najväčšiu zo 4 roč hodnôt v mriežke a vráti ju ako jednu hodnotu do výstupnej príznakovnej mapy. Toto spôsobí redukciu veľkosti príznakovnej mapy a ponecháme len najdôležitejšie príznaky.

Plne prepojené vrstvy

Ďalej po konvolučných a poolingových vrstvách, CNN zvyčajne obsahujú jednu alebo viac plne prepojených vrstiev, ktoré kombinujú príznaky extra-hované konvolučnými a poolingovými vrstvami na určenie finálnej predikcie.

Plne prepojená vrstva je zvyčajne výstupná vrstva, ktorá vracia finálnu predikciu CNN. Počet neurónov vo výstupnej vrstve závisí od úlohy, ktorú máme. Napríklad, pre klasifikáciu obrázku, výstupná vrstva môže mať jeden neurón pre každú triedu, a neurón s najvyššou aktivačnou hodnotou by predstavoval predikovanú triedu.

Tréning CNN

Tréning CNN zahŕňa prispôsobenie váh filtrov a spojení v sieti na minimalizovanie stratovej funkcie, ktorá meria rozdiel medzi predikciou a skutočným labelom. Proces trénovalia CNN môže byť rozdelený na nasledovné kroky:

Prvým krokom je vyzbierať anotovaný dataset, ktorý bude použitý na trénovanie modelu. Tento dataset by mal byť dostatočne veľký a rôznorodý aby sa naša sieť vedela generalizovať na nové obrázky. Pred tréningom siete, je zväčša nevyhnutné predspracovanie dát, aby sme sa uistili, že sú vhodné pre CNN. To môže zahŕňať prispôsobenie veľkosti obrázkov alebo augmentáciu dát aplikovaním náhodných transformácií, pre rôznorodosť datasetu a predchádzaniu overfittingu.

Ďalším krokom je rozdelenie datasetu na treningovú, validačnú a testovaciu množinu. Trénovacia množina je použitá na trénovanie CNN, validačná na vyhodnotenie CNN počas tréningu a testovacia na vyhodnotenie modelu po tréningu. Validačná množina je nápomocná pre nastavenie hyperparametrov CNN. Hyperparametre sú nastaviteľné parametre, ktoré určujú nastavenie samotného trénovacieho procesu. Testovacia množina nám poskytuje približnú schopnosť generalizácie našej siete.

Tretím krokom je návrh CNN a voľba hyperparametrov. Je veľa spôsobov ako navrhnúť CNN, ktoré majú vplyv na jej výkon, avšak lepšie ako navrhovať vlastnú CNN je zobrať nejakú existujúcu CNN, ktorá dosahuje dobré výsledky na bežných benchmarkoch. Po zvolení CNN, je dôležité správne nastavenie hyperparametrov, pre nájdenie najlepšej kardinácie pre danú úlohu. Dobré je sieť skúsať trénovať a pomaly upravovať hyperparametre.

Ďalším krokom je určenie stratovej funkcie a optimalizačného algoritmu. Stratová funkcia meria ako dobre je nás model schopný predikovať žiadaný výstup pre konkrétny vstup a jej voľba záleží na type úlohy.

Optimalizačný algoritmus (optimalizátor) je zodpovedný za prispôsobenie parametrov siete na minimalizovanie stratovej funkcie. Najbežnejší optimalizačný algoritmus je Stochastic Gradient Descent (SGD) [9]. Iný často používaný optimalizačný algoritmus je napríklad Adam [10].

Gradient Descent (Gradientový zostup) prispôsobuje parametre našej siete podľa vzorca (1.2). Vypočítaním gradientu stratovej funkcie $\frac{\partial L}{\partial w_n}$ získame smer k maximu stratovej funkcie v danom bode, teda s našimi aktualnými parametrami. My sa snažíme dosiahnuť minimum preto naše parametre upravíme v opačnom smere gradientu vynásobený krokom učenia (learning rate) η . Výpočet gradientov v našej sieti vykonávame pomocou Backpropagation [11] algoritmu, ktorý sa začína výpočtom gradientu pre výstupnú vrstvu a postupne sa propaguje späť až k vstupnej vrstve. V každej vrstve sa vypočítajú gradienty podľa chain rule of calculus, ktorá nám umožňuje vypočítať derivácie zložených funkcií. Konkrétnie, gradient stratovej funkcie vzhľadom na váhy a biasy sa vypočítajú ako súčiny derivácií stratovej funkcie vzhľadom na výstup danej vrstvy a derivácií výstupu danej vrstvy vzhľadom na váhy a biasy. Learning rate je jeden z hyperparametrov, ktorý predstavuje veľkosť kroku, ktorý spraví optimalizátor na upravenie parametrov siete. Tento optimalizačný krok iteratívne opakujeme, kým nám klesá validačná chyba. Gradient descent prispôsobuje parametre podľa všetkých tréningových príkladov.

$$w_{n+1} = w_n - \eta \frac{\partial L}{\partial w_n} \quad (1.2)$$

Stochastic gradient descent (SGD) sa lísi od Gradient Descentu tým, že neupravuje parametre vzhľadom na všetky tréningové príklady, ale len vzhľadom na niekoľko tréningových príkladov, počet týchto tréningových príkladov závisí od ďalšieho hyperparametra: veľkosť batchu. Veľkosť batchu je počet

tréningových príkladov použitých v jednej iterácii tréningu. SGD je rýchlejšie a menej výpočtovo náročné, taktiež sa pridáva šum do optimalizačného procesu, čo znižuje šancu, že algoritmus skončí v lokálnom minime.

Po trénovaní CNN je dôležité vyhodnotiť výkon na testovacej množine. Testovacia množina by mala byť dostatočne veľká aby nám ponkla spolahlivé vyhodnotenie nášho modelu a nemala by byť použitá pri tréningu.

Výkon CNN môže byť vyhodnotený pomocou rôznych metrík. Zavisí od úlohy, ktorú metriku je pre nás vhodné použiť.

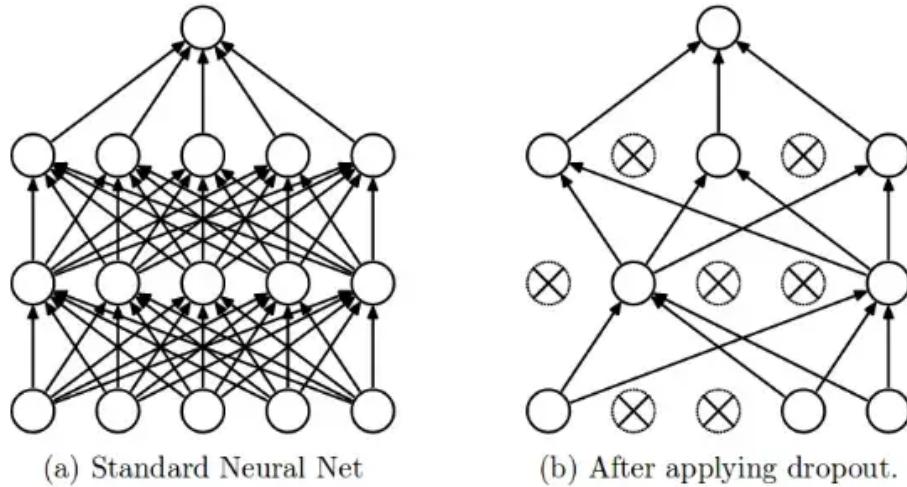
Aktivačné funkcie a regularizácia

Za účelom zavedenia nonlinearity do siete zvyčajne CNN obsahujú aktivačné funkcie v konvolučných a plne prepojených vrstvách. Najbežnejšou aktivačnou funkciou je Rectified Linear Unit (ReLU), ktorá má formu $f(x) = \max(0, x)$. Používajú sa aj iné aktivačné funkcie ako napríklad sigmoid a tanh.

Overfitting je pretrénovanie siete, to znamená, že náša sieť funguje príliš dobre na tréningových dátach, ale má to negatívny vplyv na výkon siete na nových dátach, ktoré neboli videné pri tréningu. Teda celkový výkon našej siete všeobecne na všetkých dátach sa overfittingom znižuje.

Na prevenciu overfittingu a zlepšenie generalizácie, konvolučné neurónové siete využívajú regularizačné techniky ako napríklad dropout.

Dropout [12] náhodne nepoužije nejaké percento neurónov v sieti počas každej tréningovej iterácie. ako vidíme na obrázku 1.3



Obr. 1.3: Aplikácia dropoutu

Benchmarkové metriky v objektovej detekcii

Na to aby sme vedeli vyhodnotiť presnosť modelov objektovej detekcie a porovnať ich medzi sebov potrebujeme používať rovnaké metriky. Najčastejšie používané benchmarkové metriky na určenie presnosti modelu objektovej detekcie sú mAP, mAP50, mAP75.

V objektovej detekcii, určujeme polohu objektu pomocou bounding boxu (obdĺžnik). A každému bounding boxu priradíme triedu, teda názov objektu v danej lokalite.

Pre vyhodnotenie porovnávame predikciu nášho modelu s manuálne annotovaným obrázkom. Najprv musíme spočítať počet true positive, false positive a false negative pre každý objekt zvlášť. True positive je správne detegovaný objekt. False positive je ak sme predikovali objekt tam kde nemal byť. False negative je keď sme nedetegovali objekt tam kde mal byť. Pomocou týchto hodnôt vyrátame precision (1.3) a recall (1.4)

$$Precision = \frac{\text{True positive}}{\text{True positive} + \text{False positive}} \quad (1.3)$$

$$Recall = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} \quad (1.4)$$

Na určenie, či predikovaný bounding box je dostatočne presný sa používa metrika IoU(intersection over union). Teda plocha prieniku predikcie a anotácie deleno plocha zjednotenia prieniku a anotácie (1.5). Podľa hodnoty IoU threshold určujeme, či je predikcia true positive alebo false positive. Ak je $\text{IoU} \geq \text{IoU threshold}$ pri konkrétnnej preidikcii je to true positive inak je to false positive.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (1.5)$$

Každý predikovaný bounding box obsahuje taktiež confidence score, to je skóre, ktoré popisuje ako veľmi sme si istý, že bounding box obsahuje daný objekt. Confidence threshold je hodnota, ktorá určuje minimálne confidence score na to aby bol predikovaný bounding box označený ako pozitívny.

Pre výpočet AP musíme vyrátať precision a recall pre rôzne hodnoty confidence threshold z ktorých zostojíme precision-recall krivku. Obsah pod touto krivkou je naše AP.

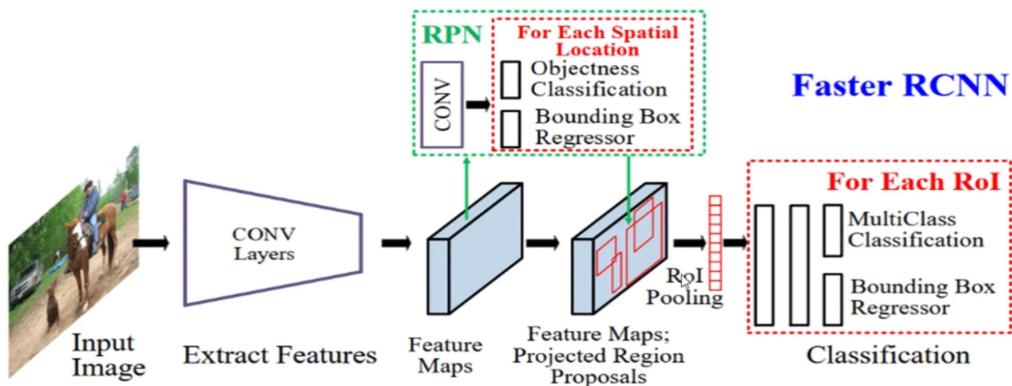
AP50 je AP pri hodnote IoU threshold = 0.5. AP75 je AP pri hodnote IoU threshold = 0.75.

Hodnotu mAP dostaneme keď spravíme priemer AP pre rôzne IoU thresholds pre každú triedu zvlášť a následne spravíme priemer týchto výsledkov pre jednotlivé triedy.

mAP50 je priemer AP50 pre jednotlivé triedy. mAP75 je priemer AP75 pre jednotlivé triedy.

Faster R-CNN

Jedna z najpoúžívanejších metód objektovej detekcie je Faster R-CNN [13]. Skladá sa z 3 hlavných častí: Backbone, RPN(Region Proposal Network) a RoI Heads. Jej architektúru vidíme na obrázku 1.4.



Obr. 1.4: Architektúra Faster R-CNN

Backbone

Backbone je kľúčovou časťou siete, ktorá zabezpečuje extrakciu príznakov zo vstupného obrázka. Faster R-CNN je pružná sieť, ktorá využíva existujúcu sieťovú architektúru, ako napríklad VGG [14], ResNet [15] alebo Inception [16], ako backbone na extrakciu príznakov.

RPN

Výstup z backbone je príznaková mapa, ktorá je vstupom pre RPN. RPN má za úlohu vygenerovať návrhy regiónov, ktoré pravdepodobne obsahujú objekt. Toto RPN dosiahne pomocou prístupu posúvneho okna a anchor boxov.

1. Anchor boxy: RPN používa množinu preddefinovaných anchor boxov.

Anchor boxy sú bounding boxy s pevne stanovenou veľkosťou, ktoré sú husto rozmiestnené po príznakovej mape. Tieto anchor boxy pokrývajú rôzne škály a pomer strán, aby zachytili objekty rôznych veľkostí a tvarov.

2. Prístup s posuvným oknom: RPN posúva malé okno zväčša 3x3 alebo 1x1 po príznakovej mape, centruje ho v každom bode príznakovej mapy a používa anchor boxy na generovanie návrhov regiónov. Posuv okna a generovanie návrhov regiónov prebiehajú súčasne pomocou konvolučnej vrstvy. Pre každý anchor box v danej lokalite RPN predikuje dve veci:
 - (a) Objectness score: RPN predikuje pravdepodobnosť, že anchor box je objekt alebo pozadie. To sa dosiahne použitím binary classification head, ktorej výstup je skóre objektu pre každý anchor box.
 - (b) Bounding Box Regression: RPN tak tiež predikuje súradnice bounding boxu, ktorý tesne obklopuje objekt, ak existuje, spojeného s anchor boxom. To sa dosiahne pomocou bounding box regression head, ktorej výstup sú jemne doladené súradnice návrhu bounding boxu na základe odchýlok od súradníc anchor boxu.
3. Non-Maximum Suppression (NMS): Po získaní Objectness score a Bounding Box Regression predikcií pre všetky anchor boxy RPN aplikuje postup Non-Maximum Suppression (NMS), aby odfiltroval nadbytočné návrhy. NMS je postup, ktorý odstraňuje prekrývajúce sa návrhy, ponecháva iba návrhy s najvyšším Objectness score a odstraňuje duplikáty alebo nadbytočné návrhy. Prekryv návrhov sa určuje pomocou IoU.
4. Doladenie návrhov: zvyšné návrhy regiónov sú doladené aplikovaním predikovanej bounding box regresie na anchor boxy.

5. Návrhy regiónov: Výstupom RPN je množina regiónov, pričom každý regón je charakterizovaný súradnicami, výškou a šírkou, taktiež má priradené skóre objektu (Objectness score).

RoI Heads

Roi Heads sa skladaju z troch častí: RoI Pooler, Box Head a Box Predictor. Vstupom pre RoI Heads je príznaková mapa (výstup z backbone) a taktiež návrhy regiónov (výstup z RPN). RoI (Region of Interest) sú návrhy regiónov, ktoré vygenerovala RPN.

1. RoI Pooler: Je zodpovedný za zmenšenie rôznych veľkostí a tvarov RoI na jednotnú veľkosť, ktorá je vhodná pre vstup do ďalších častí modelu. Vezme návrhy regiónov, ktoré sú rôznych veľkostí a tvarov a pomocou RoI Align ich transformuje na rovnakú veľkosť, zväčša je to 7x7 alebo 14x14. RoI Align je presnejšia alternatíva ako max pooling, ktorá umožňuje interpoláciu hodnôt príznakov na subpixlovú presnosť, čo zlepšuje presnosť zarovnania RoIs na príznakovú mapu. To znamená, že RoI Align dokáže zarovnať RoIs na hodnoty medzi pixelmi na príznakovnej mape.
2. Box Head: Po RoI Align fixné veľkosti reprezentácií príznakov regiónov prechádzajú cez Box Head, ktorá pozostáva z jednej alebo viacerých plne prepojených (FC) vrstiev nasledovaných aktivačnými funkciemi. Box Head je zodpovedný za ďalšie doladenie reprezentácií príznakov a generovanie predikcií pre triedy objektov a presnejšie súradnice bounding boxov.
3. Box Predictor: Výstup z Box Head ide do Box Predictora, ktorý je zodpovedný za predikciu triedy objektu a upravenie súradníc bounding

boxu, skladá sa z dvoch častí:

- (a) Box Classifier: Táto časť je zodpovedná za predikciu triedy objektu pre každý RoI. Zvyčajne sa skladá z jedného alebo viacerých plne prepojených (FC) vrstiev, nasledovaných aktivačnou funkciou softmax, ktorá generuje pravdepodobnosti pre rôzne triedy objektov.
- (b) Box Regressor: Táto časť je zodpovedná za predikciu presnejších hodnôt bounding boxu pre každý RoI. Zvyčajne sa skladá z jednej alebo viacerých plne prepojených (FC) vrstiev, nasledovaných aktivačnými funkciami ako ReLU alebo sigmoid, ktoré generujú predpovedané posuny pre bounding boxy.

1.4 Few-shot object detection(FSOD)

Problémom pri vačšine algoritmov objektovej detekcie je, že vyžadujú veľký dataset anotovaných obrázkov na tréning modelu, čo môže byť drahé a časovo náročné.

Few-shot object detection je varianta objektovej detekcie, ktorá sa snaží učiť z malého datasetu. Je to inšpirované few-shot learningom, čo je typ strojového učenia, ktorý sa učí na malých trénovacích dátach. Few-shot learning si získal v posledných rokoch veľa pozornosti, vďaka jeho schopnosti adaptovať sa novým úlohám s malým množstvom dát, čo je dôležité v prípade, že nemáme dostatok anotovaných dát.

Few-shot object detection je náročný problém, pretože model sa musí naučiť chrakteristiky objektov z malého množstva príkladov, čo je náročné vzhľadom na komplexnosť a rôznorodosť objektov. Naviac model musí rozpoznať nové triedy, ktoré neboli videné počas tréningu, čo vyžaduje dobré

rozlišovanie medzi odlišnými triedami.

Napriek náročnosti, je to dôležitý problém, pretože má potenciál výrazne znížiť počet anotovaných dát potrebných na objektovú detekciu.

1.4.1 Prístupy k FSOD

Sú viaceré prístupy, ktoré boli navrhnuté na FSOD, môžu byť zhrnuté do troch hlavných kategórií: meta-learning, transfer learning a augmentácia dát.

Meta-learning

Meta-learning je prístup, ktorý sa snaží naučiť sa učiť. Sústredí sa na trénovanie modelu, ktorý sa vie rýchlo prispôsobiť novým úlohám iba vďaka veľmi malému počtu obrázkov. Tento prístup zvyčajne zahŕňa trénovanie modelu, ktorý sa vie učiť z malého počtu dát buď použitím vonkajšej pamäti alebo optimalizačného algoritmu. Napríklad Model-Agnostic Meta-Learning (MAML) [17] algoritmus používa gradientový optimalizačný algoritmus na tréning modelu, ktorý sa vie prispôsobiť novým úlohám malým počtom aktualizácií gradientu. Algoritmus MAML bol aplikovaný na few-shot object detection pri fine-tuningu predtrénovaného modelu na objektovú detekciu na malom počte obrázkov.

Transfer learning

Transfer learning je technika pri ktorej sa využívajú parametre natrénovanej siete z jednej úlohy ako iniciálne parametre pre sieť na novú veľmi podobnú úlohu. Napríklad sieť trénovaná na veľkom datasete obrázkov zvierat by mohla byť použitá ako iniciaálna sieť pre trénovanie siete na rozpoznanie špecifického typu zvierat ako napríklad plemená psov. Použitím siete s

predtrenovanými váhami, sa naša sieť naučí rýchlejšie rozpoznávať plemená psov a stačí na to menej dát.

Augmentácia dát

Prístup augmentácie dát spočíva v rozmnožení malého množstva dát pomocou aplikovania rôznych transformácií. Zvýšením počtu dát, sa model môže naučiť robustnejšie príznaky. Pri augmentácii sa používajú transformácie ako rotácia, škálovanie, zašumenie. Používame transformácie obrazu, ktoré menia obraz, ale nemenia jeho sémantický obsah.

1.4.2 Datasetsy pre vyhodnotenie FSOD

Na vyhodnotenie výkonu FSOD algoritmov, výskumnici používajú verejne dostupné datasetsy a vyhodnocovacie metriky. Tieto datasetsy a metriky slúžia na porovnanie výkonu odlišných algoritmov a ich všeobecnosti. Najpoužívanejšie datasetsy sú:

COCO dataset [18], ktorý obsahuje 80 tried a viac ako 330 000 anotovaných obrázkov. VOC dataset [2], je to populárny dataset, ktorý obsahuje 20 tried a viac ako 11 000 obrázkov.

Tieto datasetsy sú používané ako štandard pre few-shot object detection, vyberie sa z nich niekoľko tried, ktoré sa považuju ako novel classes (triedy s malým počtom anotovaných dát), pre tieto triedy sa použije iba zopár anotovaných obrázkov (few-shot) a zvyšné triedy sa použijú na predtrénovanie modelu.

1.4.3 Aktuálne riešenia FSOD

K FSOD bolo publikovaných viacero článkov, a veľa autorov použilo iné datusety a spôsoby vyhodnotenia ich modelov. Preto je náročné ich porovnanie. Avšak, vo všeobecnosti meta-learningové prístupy zvyknú dosahovať lepšie výsledky ako transfer learning alebo prístup augmentácie dát. Hlavne preto, že meta-learningové prístupy sú špeciálne navrhnuté učiť sa z malého počtu príkladov.

Avšak, je potrebné zmieniť, že výkon few-shot object detection modelu veľmi závisí od konkrétnej implementácie, zvolených dát a zvolených metrík. A taktiež treba brať do úvahy výpočtovú a pamäťovú náročnosť.

Frustrantly simple few-shot object detection

Frustrantly simple few-shot object detection [1] je metóda, ktorú sme sa rozhodli použiť v tejto práci. Kľúčová myšlienka za touto metódou je naučiť sa detegovať objekty tréningom na množine základných tried (base classes) s veľkým počtom anotovaných obrázkov a následne spraviť fine tuning detektora na malom množstve anotovaných obrázkov z nových tried (novel classes).

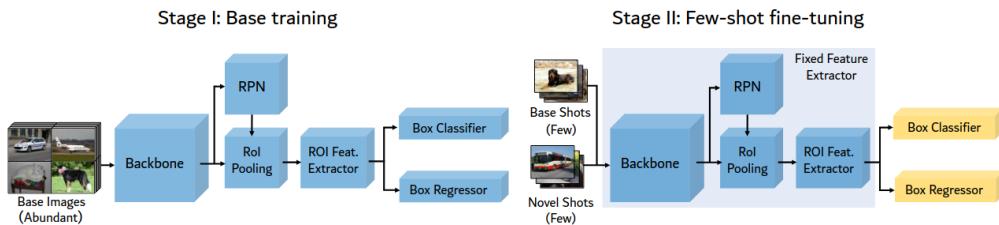
Využíva metódu Faster R-CNN [13] a spočíva v tom, že rozdelíme náš tréning na 2 etapy. V prvej etape sa vykoná base tréning na base classes klasicky cez Faster R-CNN pomocou stratovej funkcie (1.6).

$$\mathcal{L} = \mathcal{L}_{rpn} + \mathcal{L}_{cls} + \mathcal{L}_{loc} \quad (1.6)$$

Kde \mathcal{L}_{rpn} sa aplikuje na výstup z RPN na rozlíšenie popredia od pozadia, \mathcal{L}_{cls} je cross-entropy loss pre Box Classifier a \mathcal{L}_{loc} je stratová funkcia pre Box Regressor.

Následne v druhej etape tréningu (few-shot fine-tuning) vytvoríme pre každú z tried (novel aj base) malý tréningový set. Pre novel classes priradíme náhodné inicializované váhy do siete pre Box Predictor. A následne robíme fine-tuning, ale len na Box Predictor, poslednej vrstve nášho modelu. Zvyšok siete ostáva zmrazený. Použijeme rovnakú loss funkciu a 20x nižší learning rate.

Na obrázku 1.5 vidíme znázornené tieto etapy a v druhej etape vidíme žltou farbou znázornený Box Predictor, jediná časť siete, ktorá počas fine-tuningu nie je zmrazená.



Obr. 1.5: Model pre frustratingly simple few shot object detector

Kľučovým prvkom tejto metódy je oddelenie učenia sa reprezentácií príznakov a učenia sa predikovania boxov. Keďže príznaky, ktoré sme sa naučili používať na base classes môžme využiť pre novell classes.

Kapitola 2

Testovanie FSFSODT

V predošej kapitole sme si popísali ako funguje prístup metódy Frustratingly simple few shot object detection [1]. Teraz si ju prakticky vyskúšame a otestujeme pre rôzny počet trénovacích obrázkov pre novel classes. A taktiež ako sa s ňou dá reálne pracovať a ako je rýchla posledná fáza tréningu fine-tuning pri trénovaní na jednom GPU konkrétnie NVIDIA RTX 3070.

2.1 Dataset

Ako prvé bolo potreba zvoliť a pripraviť dataset na ktorom budem túto metódu testovať. Zvolil som si veľmi známy dataset PASCAL VOC [2], ktorý sa bežne používa pre porovnanie presnosti medzi rôznymi algoritmami. Tento dataset obsahuje 20 tried.

Pri few-shot object detection, potrebujeme mať dataset rozdelený na triedy ku ktorým máme veľké množstvo anotovaných dát (base classes), a na triedy ku ktorým máme malé množstvo anotovaných dát (novel classes).

Ako base classes som použil triedy: aeroplane, bicycle, boat, bottle, car, cat, chair, diningtable, dog, horse, person, pottedplant, sheep, train, tvmonitor.

nitor.

Ako novel classes som použil triedy z PASCAL VOC: bird, bus, cow, motorbike, sofa a taktiež som pridal jednu vlastnu triedu apple (anotované obrázky som stiahol z roboflow [3]) príklady anotovaných obrázkov pre túto triedu môžme vidieť na obrázku 2.1.

Kedže tvorcovia poskytli predtrénované modely, časovo a výpočtovo najnáročnejšiu prvú fázu tréningu base tréning som mohol vynechať, keďže používam rovnaké base classes dopadla by úplne rovnako a jej výstup môžem použiť pre následný fine tuning pre rôzne novel classes a aj pre rôzny počet anotovaných obrázkov pre novel classes.

Najprv treba stiahnúť PASCAL VOC dataset a umiesniť ho do priečinka `datasets` následne pridať do `datasets/VOC2007/Annotations` anotácie novo pridanej triedy, do `datasets/VOC2007/JPEGImages` obrázky tejto novej triedy a následne do `datasets/VOC2007/ImageSets/Main/trainval.txt` pridať názvy obrázkov, ktoré budeme chcieť použiť na tréning (fine tuning) a zvyšné názvy obrázkov pridáme do `datasets/VOC2007/ImageSets/Main/test.txt`, ktoré budú použité na testovanie.

Následne bolo treba pripraviť textové súbory pre každú triedu, ktoré obsahovali názvy obrázkov pre rôzny k-shot fine tuning, k-shot fine tuning znamená, že budem robiť fine tuning na k anotovaných obrázkoch pre každú novel triedu. Teda k obrázkov bude použitých na tréning a validáciu počas fine tréningu. Tieto textové súbory musia byť umiestnené v `datasets/vocsplit` pre ich generovanie môžme použiť script `datasets/prepare_voc_few_shot.py`



Obr. 2.1: Príklady anotovaných obrázkov pre triedu apple

2.2 Tréning

Po stiahnutí predtrénovaného modelu na 15 base triedach bolo treba inicializovať hodnoty pre nové triedy v Box Classifier. Box Classifier mal pôvodne výstup pravdepodobnostného rozloženia pre 15 tried, ale teraz máme 21 tried, takže treba váhy a biasy pre pôvodné triedy ponechať a inicializovať náhodné hodnoty pre nové triedy.

Nasleduje na rad fine-tuning. Keďže autori použili na tréning 8 16GB gpu, musel som prispôsobiť konfiguračné parametre tréningu aby som si vystačil s pamäťou 1 8GB gpu.

Znížil som batch size z 16 na 2, learning rate z 0.001 na 0.000125. A zvýšil som step z 3000 na 24000 a max_iter z 4000 na 32000. Teda batch size a learning rate som 8-násobne znížil a step(počet iterácií, po ktorých sa learning rate desaťnásobne zníži) a max_iter(počet itérácií) som 8-násobne zvýšil. Pri každej k-shot detekcii máme odlišný step a max_iter. Čím viac trénovacích obrázkov, tead čím vyššie k, tým je potrebné trénovať dlhšie pre optimálne výsledky a preto treba zvýšiť tieto dva parametre.

Ako optimalizátor budeme používať Stochastic gradient descent s mo-

mentom(SGDM), je to modifikácia SGD, ktorá zahŕňa využitie momenta z predošlého kroku pre zrýchlenie konvergencie a prekonanie lokálneho minima. Aktualizácia váh po jednom kroku pomocou SGDM je vyjadrená rovnicami (2.1) a (2.2). Pričom $v_0 = 0$ a ρ je koeficient momenta, my budeme používať $\rho = 0.9$.

$$v_{n+1} = \rho v_t + \frac{\partial L}{\partial w_n} \quad (2.1)$$

$$w_{n+1} = w_n - \eta v_{n+1} \quad (2.2)$$

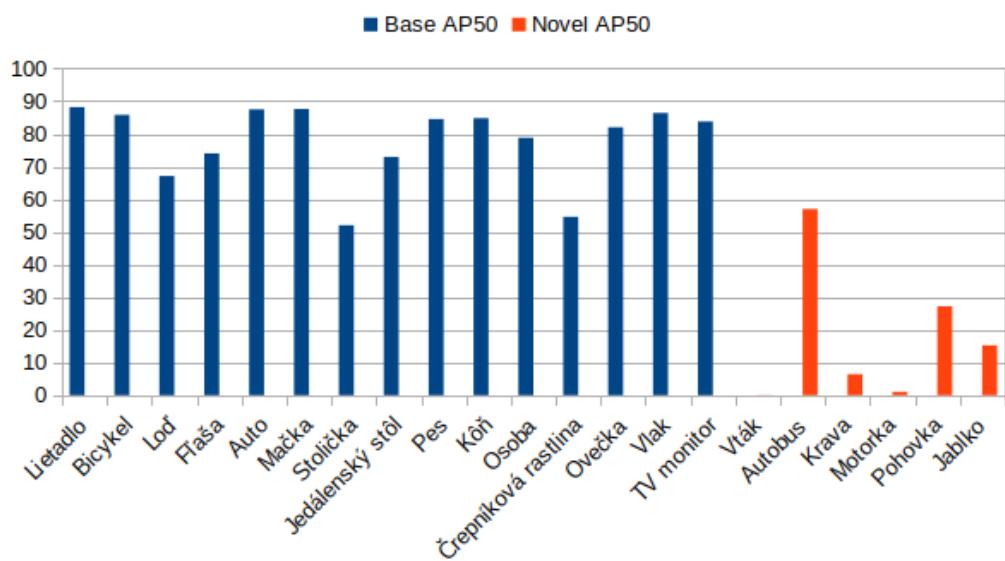
2.3 Testovanie rýchlosi a presnosti modelu pri zmene počtu trénovacích obrázkov novel tried

Teraz si otestujeme ako rýchly bude tréning a následne akú presnosť bude dosahovať model pri rôznom počte trénovacích obrázkov novel tried. Trénovacie obrázky pre každú triedu sú vybrané náhodne.

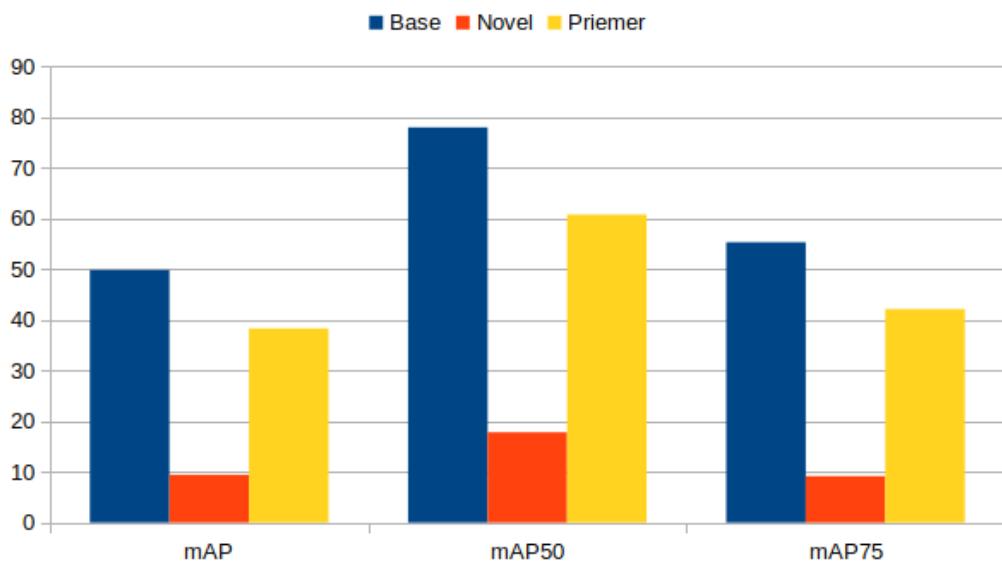
2.3.1 1-shot detekcia

Najprv som sa rozhodol algoritmus otestovať pre 1-shot detekciu, teda druhá fáza tréningu fine-tuning bude prebiehať len na 1 obrázku z každej triedy (novel aj base) teda na 21 obrázkoch, ktoré budú taktiež resizované tak že kratšia hrana obrázku bude resizovaná na tieto dĺžky: 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800 a dlhšia hrana bude prispôsobená tak aby bol zachovaný pomer hrán, s tým že maximálna veľkosť dlhšej hrany je 1333 a teda ak by mal náš resize presiahnuť túto veľkosť nastaví sa dlhšia hrana obrázka na 1333 a kratšia hrana sa prispôsobí na veľkosť aby sa zachoval rovnaký pomer strán ako pri originálnom obrázku.

Celkový čas tréningu: 55 minút a 6 sekúnd, na obrázku 2.2 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti nie je rovnomerné a pre niektoré triedy je presnosť takmer nulová. Na obrázku 2.3 a v tabuľke 2.1 vidíme priemernú presnosť nášho modelu.



Obr. 2.2: AP50 na testovacích dátach pre jednotlivé triedy pre 1-shot detekciu.



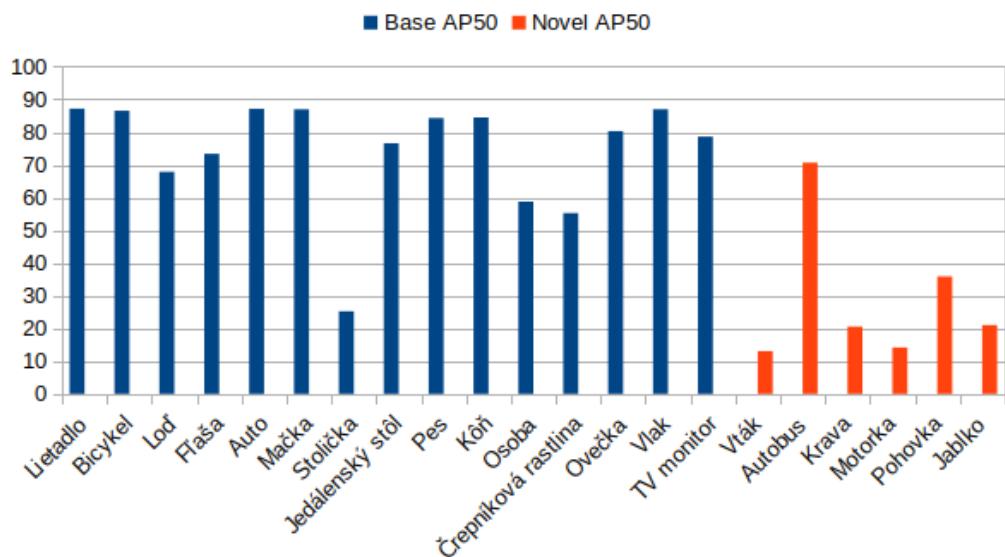
Obr. 2.3: Graf presností na testovacích dátach pre 1-shot detekciu.

Presnosť	Base	Novel	Priemer
mAP	49.858	9.41	38.301
mAP50	78.004	17.812	60.806
mAP75	55.335	9.142	42.137

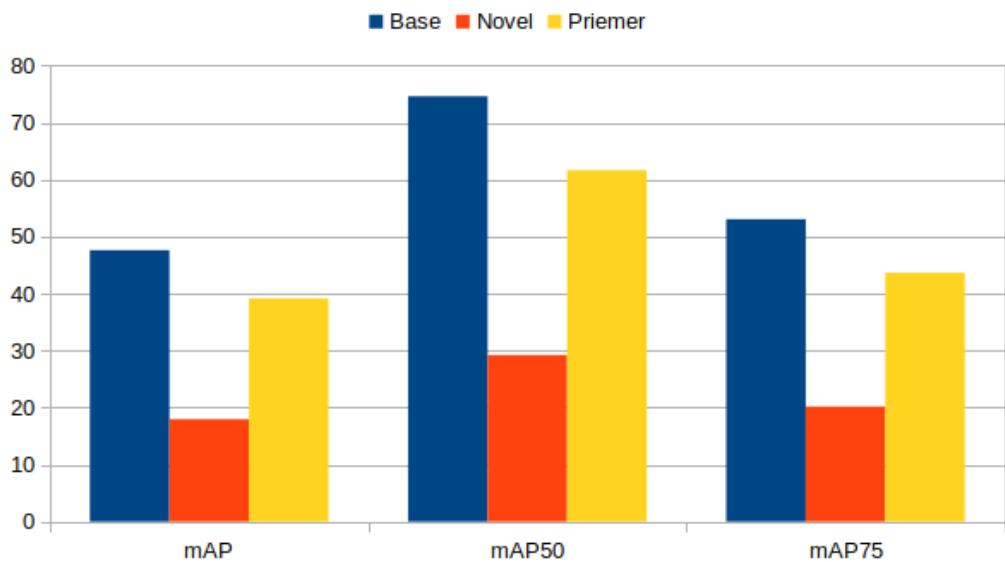
Tabuľka 2.1: Tabuľka presností na testovacích dátach pre 1-shot deketcii.

2.3.2 2-shot detekcia

Celkový čas tréningu: 1 hodina, 51 minút a 41 sekúnd, na obrázku 2.4 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti sa zlepšila oproti 1-shot detekcii a už dokážeme rozpoznať každý objekt aspoň s nejakou presnosťou. Na obrázku 2.5 a v tabuľke 2.2 vidíme priemernú presnosť nášho modelu.



Obr. 2.4: AP50 na testovacích dátach pre jednotlivé triedy pre 2-shot detekciu.



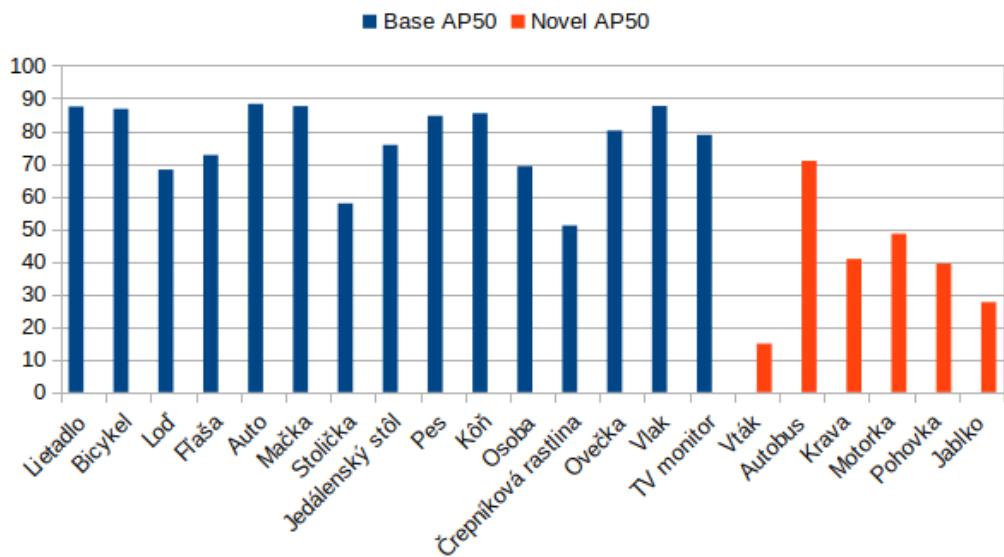
Obr. 2.5: Graf presnosťí na testovacích dátach pre 2-shot detekciu.

Presnosť	Base	Novel	Priemer
mAP	47.623	17.947	39.144
mAP50	74.623	29.218	61.65
mAP75	53.055	20.182	43.663

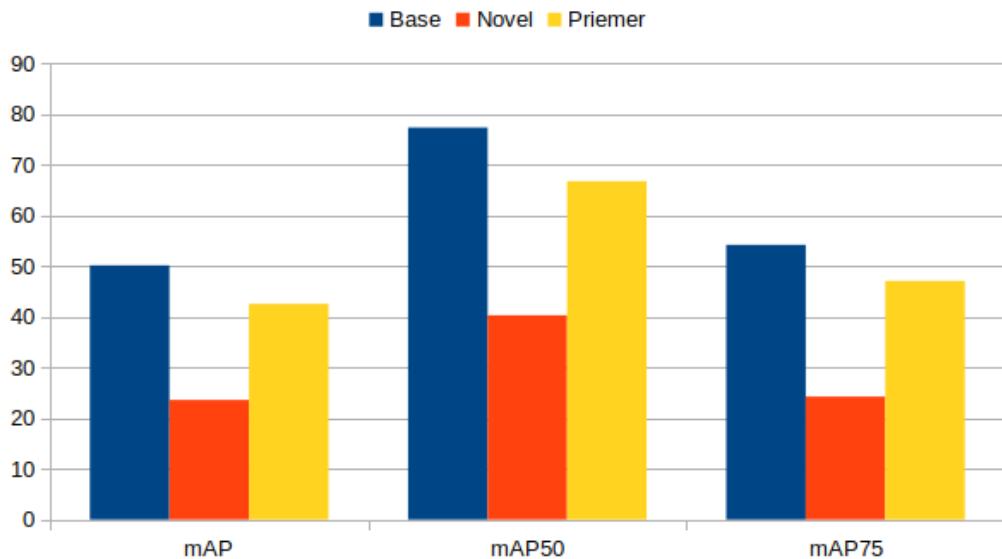
Tabuľka 2.2: Tabuľka presnosťí na testovacích dátach pre 2-shot deketcu.

2.3.3 3-shot detekcia

Celkový čas tréningu: 2 hodiny, 46 minút a 44 sekúnd, na obrázku 2.6 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti sa opäť zlepšilo, ale pre triedu vták je stále dosť nízka presnosť. Na obrázku 2.7 a v tabuľke 2.3 vidíme priemernú presnosť nášho modelu.



Obr. 2.6: AP50 na testovacích dátach pre jednotlivé triedy pre 3-shot detekciu.



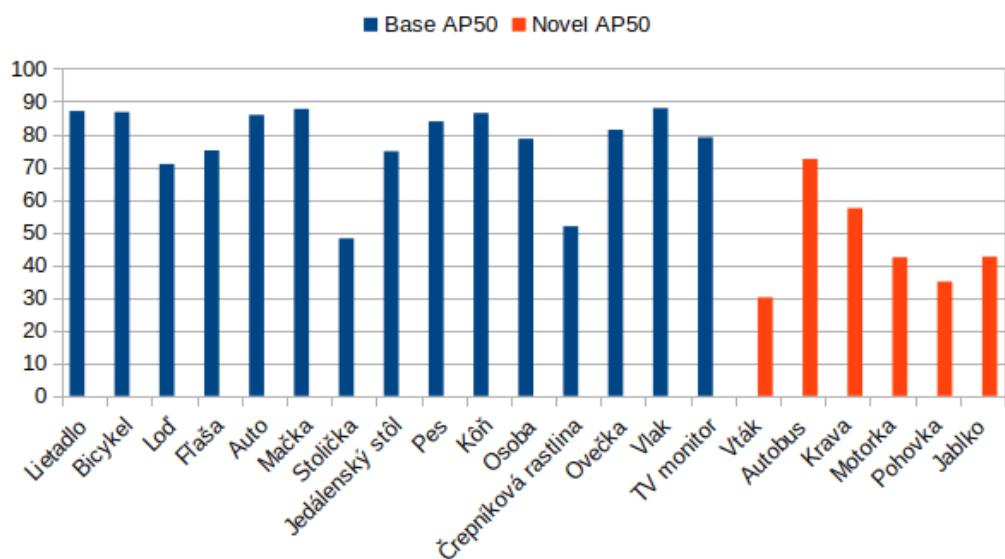
Obr. 2.7: Graf presností na testovacích dátach pre 3-shot detekciu.

Presnosť	Base	Novel	Priemer
mAP	50.173	23.598	42.58
mAP50	77.393	40.309	66.798
mAP75	54.235	24.273	47.103

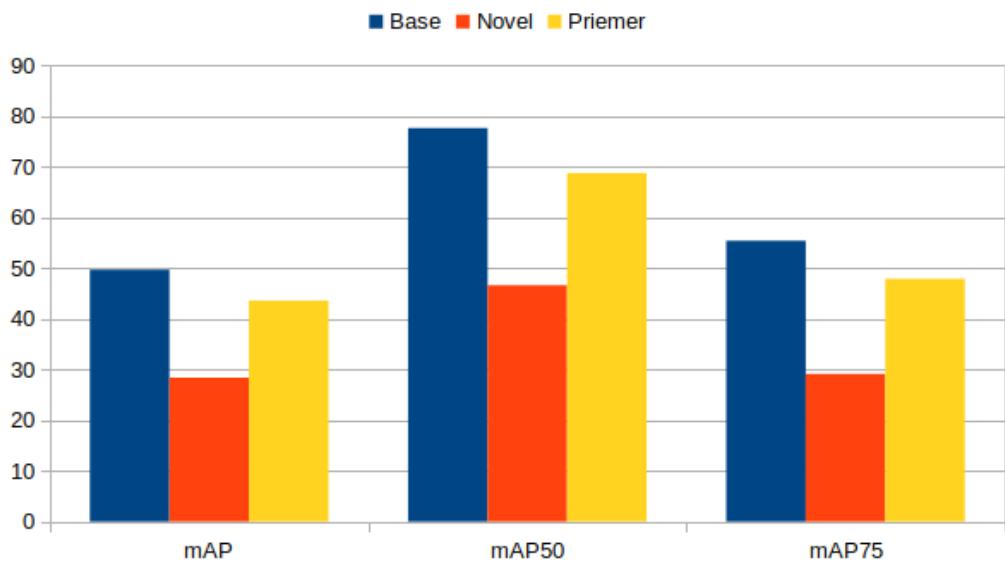
Tabuľka 2.3: Tabuľka presností na testovacích dátach pre 3-shot deketcu.

2.3.4 5-shot detekcia

Celkový čas tréningu: 4 hodiny, 35 minút a 25 sekúnd. Ako vidíme trénin-gový čas nám veľmi rastie pri zvyšovaní počtu trénovacích obrázkov, keďže taktiež treba zvyšovať počet iterácií, aby sme dosiahli optimálne výsledky. Na obrázku 2.8 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti sa zvyšovaním počtu trénovacích obrázkov stále zvyšuje a tentokrát už máme pre každú triedu AP50 nad 30. Na obrázku 2.9 a 2.4 vidíme priemernú presnosť nášho modelu.



Obr. 2.8: AP50 na testovacích dátach pre jednotlivé triedy pre 5-shot detekciu.



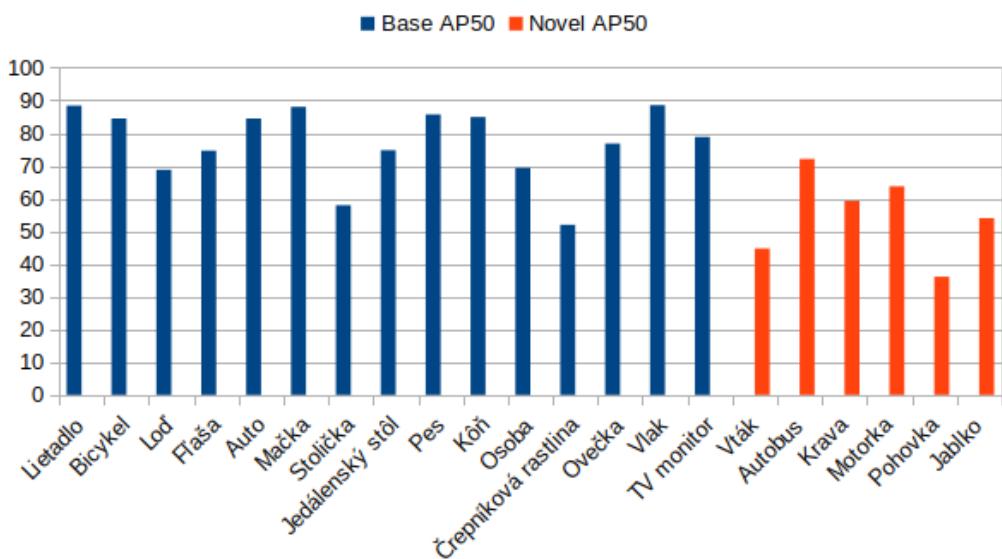
Obr. 2.9: Graf presnosťí na testovacích dátach pre 5-shot detekciu.

Presnosť	Base	Novel	Priemer
mAP	49.71	28.395	43.62
mAP50	77.677	46.63	68.806
mAP75	55.442	29.132	47.925

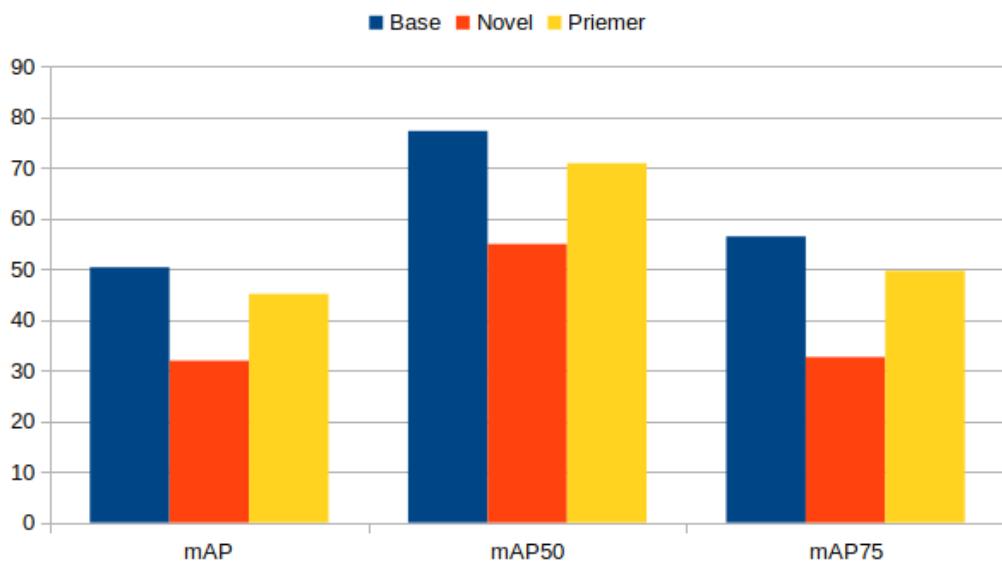
Tabuľka 2.4: Tabuľka presnosťí na testovacích dátach pre 5-shot detekciu.

2.3.5 10-shot detekcia

Celkový čas tréningu: 9 hodín, 11 minút a 4 sekundy. Na obrázku 2.10 vidíme presnosť AP50 pre jednotlivé triedy. Na obrázku 2.11 a v tabuľke 2.5 vidíme priemernú presnosť nášho modelu.



Obr. 2.10: AP50 na testovacích dátach pre jednotlivé triedy pre 10-shot detekciu.



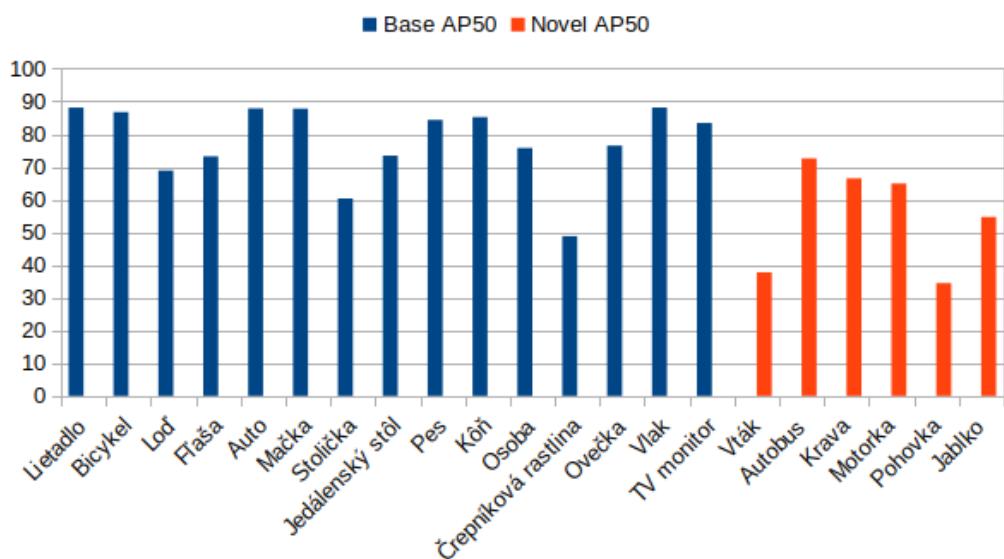
Obr. 2.11: Graf presností na testovacích dátach pre 10-shot detekciu.

Presnosť	Base	Novel	Priemer
mAP	50.394	31.946	45.123
mAP50	77.305	54.99	70.93
mAP75	56.49	32.666	49.683

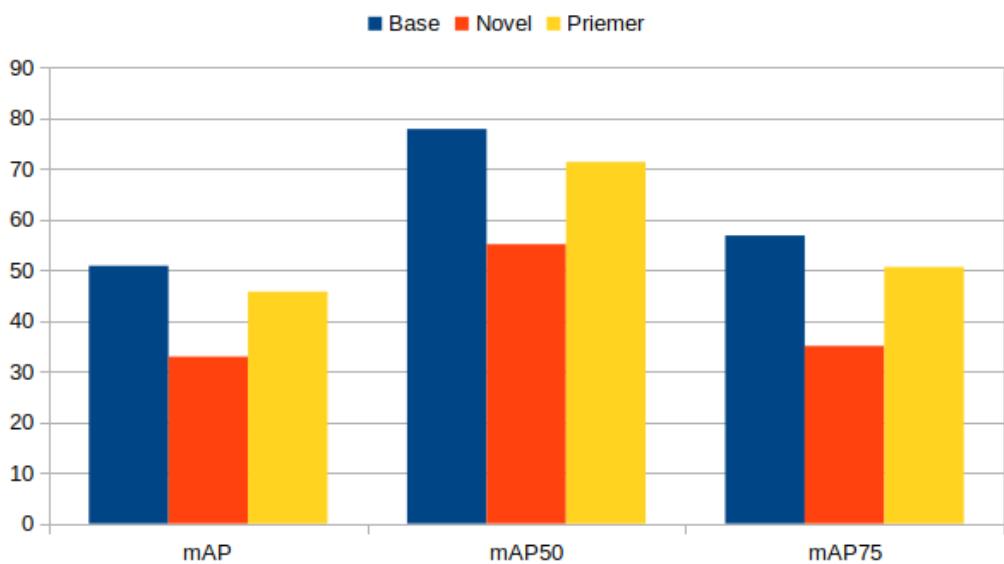
Tabuľka 2.5: Tabuľka presností na testovacích dátach pre 10-shot detekciu.

2.3.6 15-shot detektia

Celkový čas tréningu: 13 hodín, 56 minút a 37 sekúnd. Na obrázku 2.12 vidíme presnosť AP50 pre jednotlivé triedy. Na obrázku 2.13 a 2.6 vidíme priemernú presnosť nášho modelu. Presnosť nášho modelu sa len minimálne zvýšila oproti 10-shot detektii a pritom dĺžka tréningu sa zvýšila približne o 50 percent.



Obr. 2.12: AP50 na testovacích dátach pre jednotlivé triedy pre 15-shot detekciu.



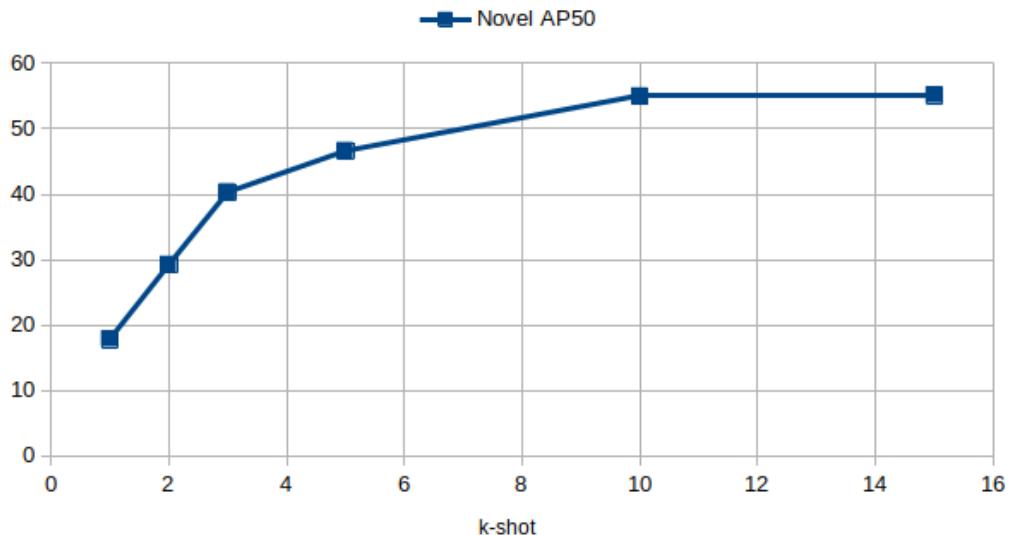
Obr. 2.13: Graf presnosťí na testovacích dátach pre 15-shot detekciu.

Presnosť	Base	Novel	Priemer
mAP	50.891	32.951	45.766
mAP50	77.887	55.142	71.388
mAP75	56.88	35.069	50.648

Tabuľka 2.6: Tabuľka presnosťí na testovacích dátach pre 15-shot detekciu.

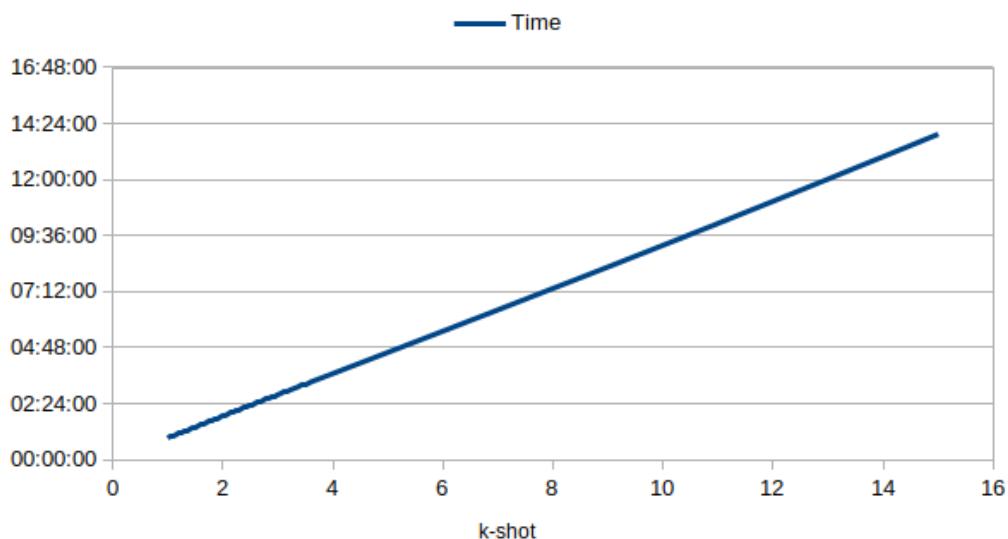
2.3.7 Vyhodnotenie

Teraz vyhodnotíme rôzne vlastnosti fine-tuningu vzhľadom k počtu trénovacích obrázkov.



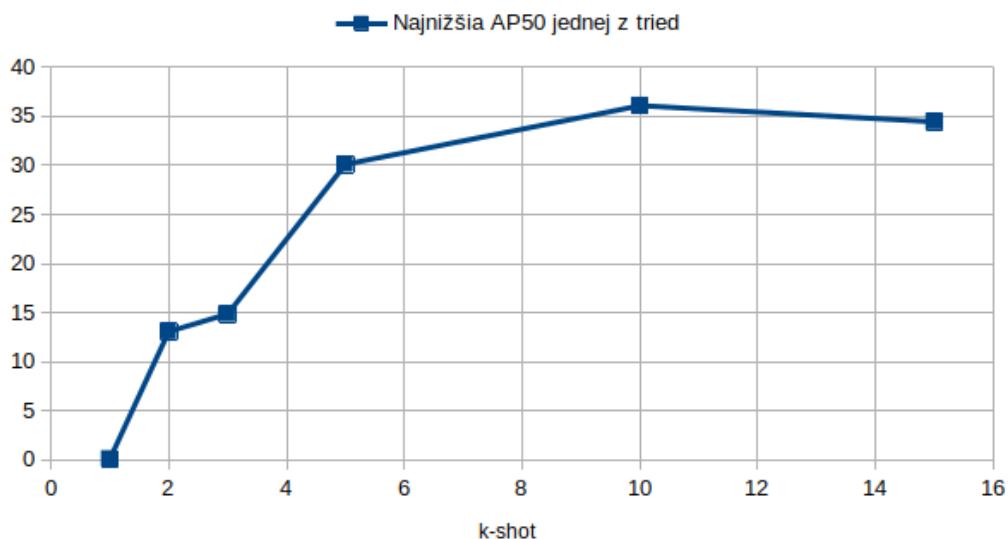
Obr. 2.14: Novel AP50 na testovacích dátach vzhľadom k počtu trénovacích obrázkov.

Na obrázku 2.14 vidíme presnosť novel tried vzhľadom k počtu trénovacích obrázkov. Vidíme, že presnosť nám stále stúpa, ale po 10tich trénovacích obrázkoch nám presnosť stúpa už iba minimálne. Pri pätnástich trénovacích obrázkoch nám oproti desiatim stúpla presnosť iba o zanedbatelných 0.153.



Obr. 2.15: Čas tréningu vzľadom k počtu trénovacích obrázkov.

Na ďalšom obrázku 2.15 vidíme ako sa nám mení čas tréningu, vzhlľadom k počtu trénovacích obrázkov. Vidíme lineárnu funkciu, čas tréningu nám priamoúmerne stúpa s počtom trénovacích obrázkov.



Obr. 2.16: Najnižšia AP50 na testovacích dátach vzhľadom k počtu trénovačích obrázkov.

Ďalšia zaujímava metrika na obrázku 2.16 nám ukazuje presnosť triedy s najmenšou presnosťou vzhľadom k počtu trénovacích obrázkov. Vidíme, že pri 1-shot detekcii jedna z tried má takmer nulovú presnosť, čo nie je úplne ideálne, keď je pre nás dôležité rozpoznávať všetky triedy. Tento graf nám teda zobrazuje garanciu najnižej presnosti pre každú z tried bez ohľadu na priemernú prenosť. Vidíme, že od 5-shot learningu máme celkom solidnú garanciu aspon 30 percent AP50 pre každú triedu. Pri 15-shot detekcii sa nám najnižšia presnosť dokonca o trochu zníži oproti 10-shot detekcii, napriek jemnému zvýšeniu priemernej presnosti.

Kapitola 3

Zrýchlenie FSFSOD

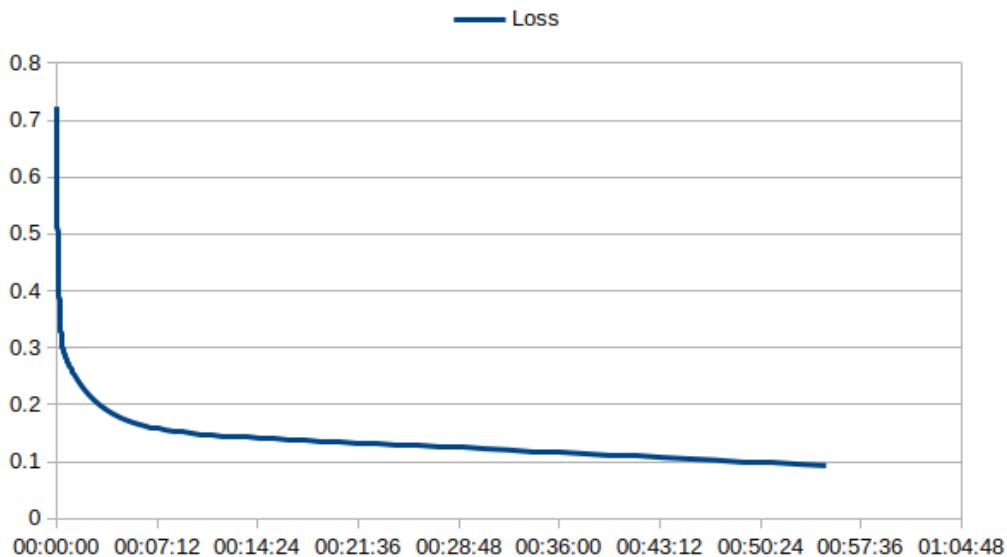
Ako sme videli v predošej kapitole fine tuning Frustratingly Simple Few Shot Object Detection [1] trvá pomere dlho a v žiadnom prípade sa nedá použiť v real-time na učenie nových objektov. Preto sa v tejto kapitole pozrieme na to ako by sa dal zrýchliť.

3.1 Zrýchlenie pomocou zmeny parametrov

Pri pokuse o zrýchlenie fine-tuningu mi ako prvé napadlo zmena trénovacích parametrov. Parametre ktoré majú vplyv na dĺžku tréningu sú batch-size a počet trénovacích iterácií. Pri batch-size sme obmedzený pamäťou našej grafickej karty a maximálny batch size, ktorý zvladne je 2, takže s týmto parametrom nepohneme. Avšak všimol som si, že počas fine-tuningu sa náš model trénuje na začiatku veľmi rýchlo a časom sa veľmi spomaluje.

Najobjektívnejšiu metriku, ktorú máme počas tréningu je total loss. Vyskúšame ako sa mení počas 1-shot fine-tuningu. Na začiatku má hodnotu 0.7237 a na konci 0.09278. Behom prvej minúty klesne približne na 0.25. Po piatich minútach má hodnotu 0.17. Po 20 minútach je na hodnote 0.13 a

ďalších 35minút veľmi pomaly klesá až na finálnu hodnotu 0.09278. Celý priebeh vidíme na obrázku 3.1.

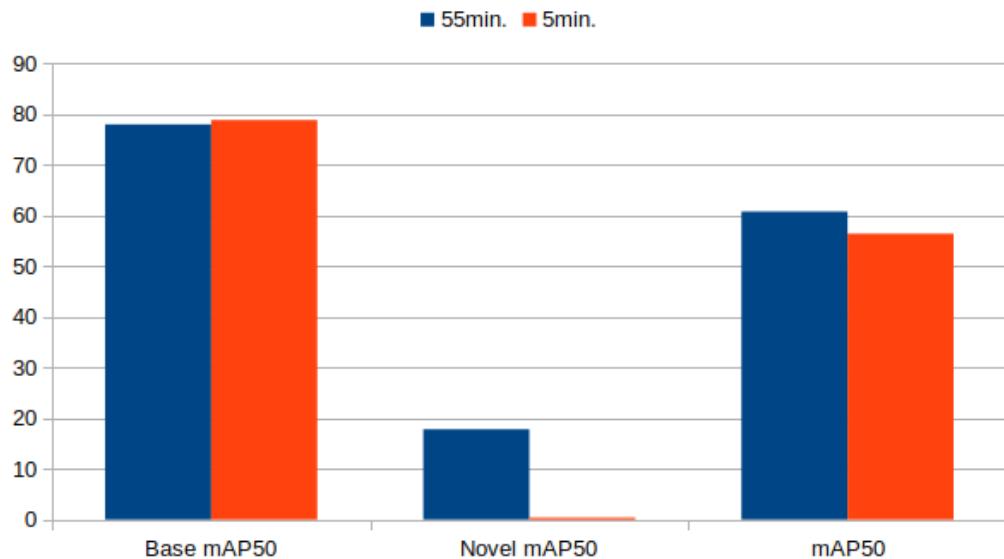


Obr. 3.1: Celková strata počas 1-shot fine tuningu

Už po piatich minútach máme náš model podľa tejto metriky výrazne natrénovaný. Pozrime sa teda na ostatné metriky presnosti po 5 minútach tréningu. Vyskúšame teda najprv znížiť počet iterácií, tak aby tréning trval približne 5 minút a teda znížime počet iterácií z 32 000 na 3 200 a uvidíme ako veľmi nám klesne presnosť pri znížení trénovacieho času o desatinu.

Presnosť	Base	Novel	Priemer
mAP	51.505	0.114	36.822
mAP50	78.865	0.379	56.44
mAP75	57.286	0	40.919

Tabuľka 3.1: Tabuľka presností na testovacích dátach po 5 min. tréningu pre 1-shot detekciu.

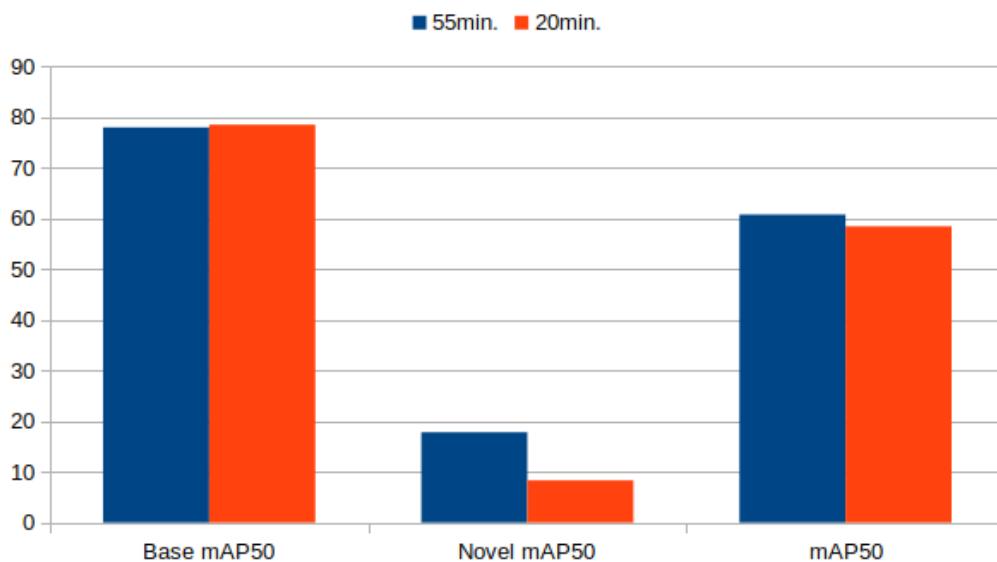


Obr. 3.2: Porovnanie mAP50 na testovacích dátach 5 min. tréningu s plným tréningom.

Ako vidíme v tabuľke 3.1 a na obrázku 3.2 priemerná presnosť síce ostala celkom vysoká a presnosť pre base classes je dokonca vyššia, ale pre nás je najdôležitejšia presnosť pre novel classes a tá má veľmi nízke hodnoty, takže takéto skrátenie tréningu nám prinieslo veľmi zlé výsledky. Vyskúšame teda ešte skrátiť tréning na 20 minút a teda znížime počet iterácií z pôvodných 32 000 na 12 000.

Presnosť	Base	Novel	Priemer
mAP	50.479	4.092	37.226
mAP50	78.541	8.326	58.479
mAP75	56.248	3.701	41.235

Tabuľka 3.2: Tabuľka presností na testovacích dátach po 20 min. tréningu pre 1-shot detekciu.



Obr. 3.3: Porovnanie mAP50 na testovacích dátach 20 min. tréningu s plným tréningom

V tabuľke 3.2 vidíme výsledky po 20 minútach a na obrázku 3.3 porovnanie presnosti s plným tréningom. Priemerná presnosť sa oproti 5 minútovému tréningu zvýšila len trochu a vidíme taktiež, že sa nám zvýšila presnosť pre novel classes, ale stále je viac ako o polovicu nižšia ako pri plnom čase tréningu.

3.2 Zrýchlenie pomocou zapamätania si výstupu zo zmrazených vrstiev

Predošlý pokus o zrýchlenie neboli veľmi úspešný, avšak zistil som, že počas tréningu prechádza každý batch vždy celou sieťou, napriek tomu, že pri fine tuningu je takmer celá sieť zmrazená okrem poslednej vrstvy (Box Predictor). Vyskúšame teda prejsť zmrazenou časťou siete pre každý vstup len raz a zapamätať si výstup z tejto časti siete pre každý vstup a následne používať len

tento predom vypočítaný výstup zo zmrazenej časti siete pre zvyšok tréningu, čo by malo výrazne urýchliť prechod sieťou.

3.2.1 Zapamätanie si výstupu z backbone

Ako sme si popísali v 3. kapitole naša sieť sa skladá z 3 hlavných častí: Backbone, RPN a Roi Heads. Drivivú väčšinu našej siete tvorí Backbone, ktorej výstup ide do RPN a Roi Heads. Do Roi Heads ide taktiež výstup z RPN. V našom prípade pri fine-tuningu sú zmrazené všetky vrstvy okrem poslednej - Box Predictor. Keďže vstupom pre Box Predictor je len výstup z Box Head, potrebujeme si zapamätať výstup z Box Head. Keďže, výstupom Box Predicota je len zoznam s dvomi hodnotami:

1. Class Prediction: pravdepodobnostné rozloženie medzi triedami objektov
2. Bounding Box Regression: posun pôvodných súradníc bounding boxu z RPN

potrebujeme si taktiež zapamätať informácie o návrhoch regiónov, teda výstup z RPN, pre následné vypočítanie straty kvôli tréningu.

Keďže zapamätanie si výstupu pre každý obrázok z Box Head vrstvy, ktorá je súčasťou Roi Heads a výstup návrhov regiónov, ktoré sú výstupom z RPN, je implementačne náročnejšie. Vyskúšame najprv zapamietanie si výstupu z Backbone, ktorý aj tak tvori 90% siete a mal by priniesť výrazné zrýchlenie. Ak by nám to neprinieslo očakávané zrýchlenie, nebude mať zmysel posúvať túto myšlienku ďalej a zapamätať si výstup priamo z Box Head a RPN pre každý obrázok, čo by malo maximalizovať rýchlosť tréningu.

Keďže, počas tréningu používame batch size veľkosti 2 a obrázky, ktoré sa berú náhodne z tréningovej množiny sú rôznej veľkosti a taktiež sú nie-

koľko krát resizované, tak aby bol zachovaný pomer strán. Tieto dva vybrané obrázky sú prispôsobené na rovnakú veľkosť pomocou paddingu aby mohli byť reprezentované v jednom tenzore, pred tým ako prejdu Backbonom.

Týmto sa nám zapamätávanie výstupov veľmi komplikuje, pretože to aký padding bude pridaný na obrázok zavisí od rozmeru obrázka s ktorým je v batchi. A na to aby sme boli presný museli by sme si pamätať výstup pre každú dvojicu obrázkov zvlášť a to určite nechceme, keďže už pri 1-shot detekcii pri 21 triedach máme 21 obrázkov a každý je 11x resizovaný, čiže 231 obrázkov, teda $231 \times 230 = 53\ 130$ rôznych dvojíc obrázkov a rôznych príznakových máp.

Máme niekoľko možností ako to riešiť, vyskúšame si každú z nich:

1. Padding pre všetky obrázky na rovnakú veľkosť

Môžme zistiť aké rozmery má najväčší batch a nastaviť padding pre každý obrázok na tieto rozmery. Všetky obrázky by tak mali rovnaký rozmer a taktiež aj ich príznaková mapa. Čiže by sme mohli vyrátať príznakovú mapu pre každý obrázok vo všetkých veľkostiach a jednoducho by sme potom takéto dve príznakové mapy spojili do batchu. Problém je v tom, že obrázky budú obsahovať zbytočne veľa paddingu, a všetky obrázky budú príliš veľke, čo taktiež veľmi spomalí nás tréning. Pri vyskúšaní tohto postupu som zistil, že tieto veľké obrázky zaberali príliš pamäte a nestačila na to ani moja grafická karta.

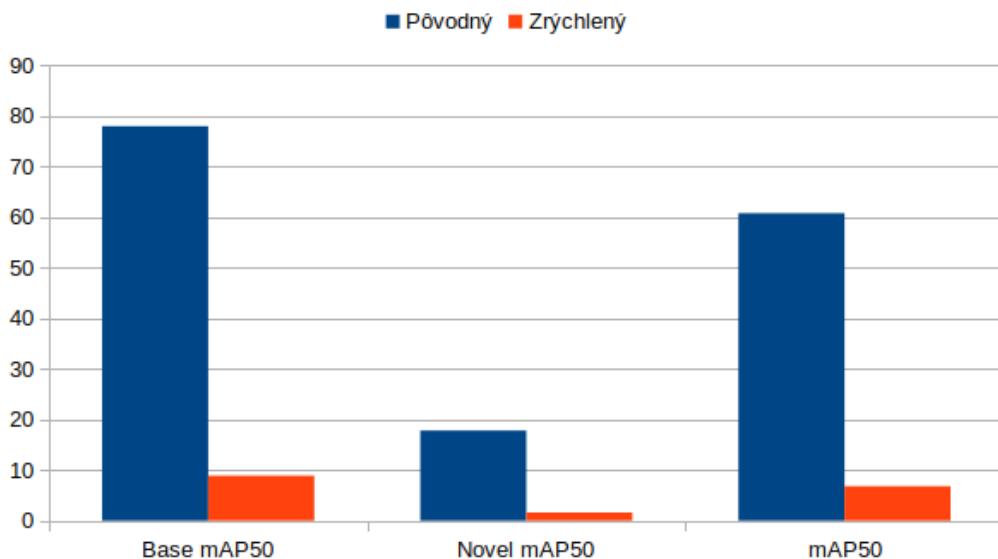
2. Prispôsobiť všetky obrázky na rovnakú veľkosť

Ďalší spôsob, je nastaviť všetkým obrázkom rovnakú veľkosť a to spôsobom, že im nastavím veľkosť dlhšej hrany na veľkosť 480, kratšia hrana sa prispôsobí aby sa zachoval pomer strán a následne je obrázok doplnený paddingom

tak aby mal rozmer 480x480.

Presnosť	Base	Novel	Priemer
mAP	4.882	0.788	3.712
mAP50	8.875	1.6	6.797
mAP75	4.684	0.874	3.596

Tabuľka 3.3: Tabuľka presnosťí na testovacích dátach pre prispôsobené obrázky na rovnakú veľkosť pri uložení výstupu z Backbone pri 1-shot detekcii.



Obr. 3.4: Porovnanie presnosťí pôvodného a zrýchleného algoritmu pomocou zapamätania výstupu z Backbone a prispôsobenia obrázkov na rovnakú veľkosť

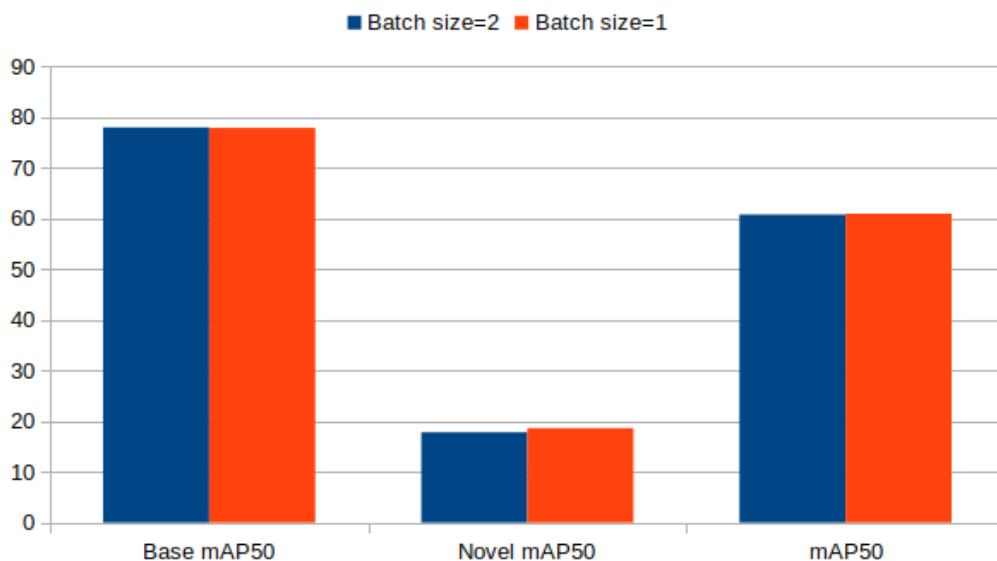
Tréning sa nám sice podarilo výrazne zrýchliť, trval len 13 minút a 27 sekúnd, avšak ako vidíme v tabuľke 3.3 a na obrázku 3.4 v porovnaní s presnosťou pôvodného algoritmu naša presnosť veľmi klesla.

Takúto nízku presnosť máme hlavne preto, že náš model je testovaný na obrázkoch s veľkosťou dlhšej hrany rovnej 800. Skúšili sme teda zmeniť veľkosť testovacích obrázkov na veľkosť dlhšej hrany rovnej 480. Dosiahli sme tak podobné výsledky ako pri pôvodnom tréningu, avšak tento výsledok

nie je dostatočne dôveryhodný keďže sme si prispôsobili veľkosť testovacích obrázkov ako sme potrebovali a pri reálnom používaní by bol model menej presný.

3. Použiť batch size = 1

Doteraz sme robili tréning pri batch size veľkosti 2, vyskúšame ako bude algoritmus fungovať pri batch size 1. Ak bude dosahovať dobré výsledky, môže nám to uľahčiť zrýchlenie tréningu. Otestujeme to na 1-shot fine-tuningu, pretože prebieha najrýchlejšie. Prispôsobíme teda trénovacie parametre batch size zmenšíme o polovicu na 1, learning rate taktiež znížime o polovicu z 0.000125 na 0.0000625, zdvojnásobíme krok z 24 000 na 48 000 a taktiež zdvojnásobíme počet iterácií na 64 000.



Obr. 3.5: Porovnanie presnosti pri batch size = 1 a batch size = 2'

Na obrázku 3.5 vidíme, že tréning pri batch size veľkosti 1 dosahuje veľmi podobné výsledky ako pri tréningu s batch size 2, avšak tréning bol menej stabilný a celková strata (total loss) počas tréningu s batch size 1 skákala z

iterácie na iteráciu s oveľa väčším rozptylom. Čo môže spôsobiť rozptyl vo výsledkoch pri viacerých tréningoch. Každý tréning sa môže podať inak a pri batch size 1 máme nižšiu konzistenciu.

Toto sa nám ale podarilo vyriešiť pomocou zmeny Checkpointera. Checkpointer zaznamená aktuálny stav modelu a uloží ho do súboru. Predtým sme používali PeriodicCheckpointer, ktorý vytvoril checkpoint periodicky po rovnakom počte iterácií. Počet iterácií po ktorých sa spraví checkpoint je nastaviteľný a mali sme ho nastavený na 500. Použijeme však BestCheckpointer, ktorý spraví checkpoint ak bola total loss nižšia v aktuálnej iterácii ako v poslednom checkpointe. Nastavíme ho aby kontroloval každých 100 iterácií. Následne ako finálny model berieme posledný checkpoint.

Tento tréning s batch size 1 trval dokonca o 4 minúty kratšie. Zrejmä preto, že spojiť 2 obrázky rôznych rozmerov do jedného batchu je výpočtovo náročnejšie. A keďže osahuje veľmi dobré výsledky, stojí za to ho využiť v tomto princípe.

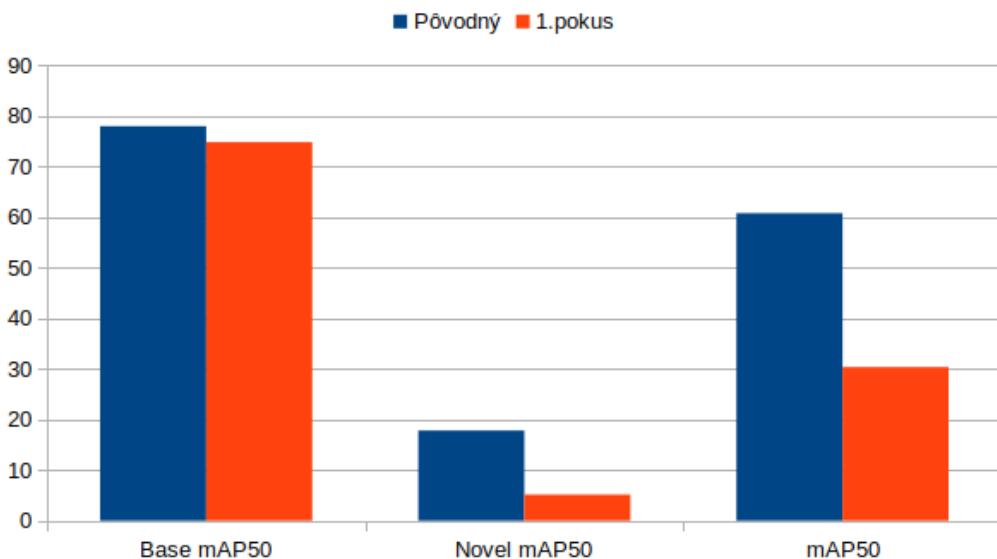
1.pokus

Vyskúšame si teda zapamätať výstup z Backbone pre každý obrázok a následne už Backbonom prechádzať nebudeme. Uvidíme ako to zrýchly náš tréning a aký výsledok dosiahneme.

Čas nášho tréningu výrazne klesol, tréning trval iba 20 min. oproti pôvodným 55 min. Avšak taktiež klesla naša presnosť ako vidíme na obrázku 3.6 v porovnaní s pôvodným tréningom. Hlavným dôvodom poklesu presnosti bude vynechávanie augmentácií počas tréningu, keďže pre každý obrázok si zapamätáme príznaky iba pre jednu transformáciu toho obrázku. Pôvodne počas sa počas tréningu aplikovali na obrázok rôzne augmentáciu a to zmena veľkosti a taktiež náhodné horizontálne otočenie.

Presnosť	Base	Novel	Priemer
mAP	41.895	1.52	30.359
mAP50	74.852	5.143	54.935
mAP75	41.896	0.347	30.025

Tabuľka 3.4: Tabuľka presnosťí na testovacích dátach 1. pokusu o zrýchlenie pri batch size = 1.



Obr. 3.6: Graf porovnania mAP50 na testovacích dátach pôvodného algoritmu a 1. pokusu o zrýchlenie pri batch size = 1.

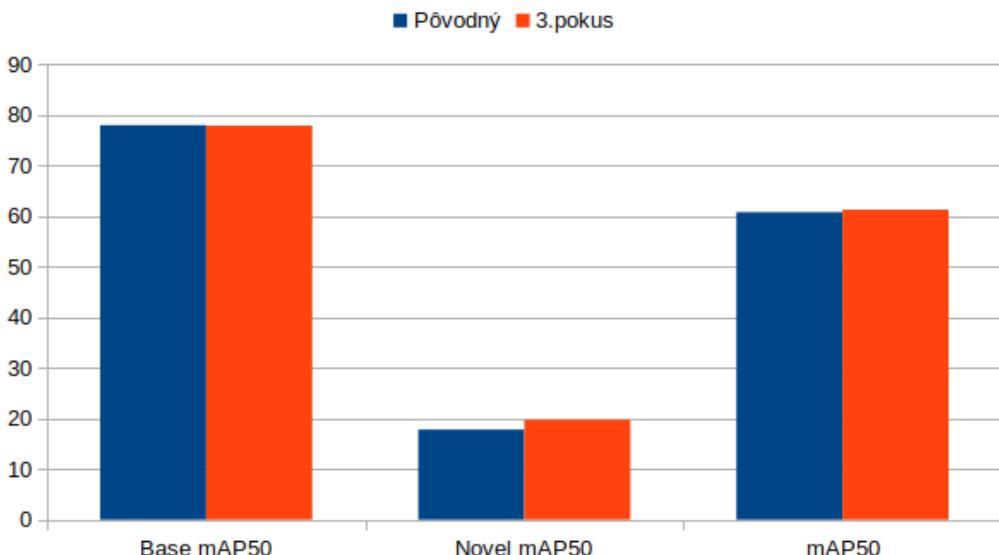
2.pokus

Pre zvýšenie našej presnosti si vyskúšame zapamätať každý obrázok vo všetkých jeho šírkach. Pokúsime naše príznaky pre každý obrázok v každej veľkosti zapísat do súboru a následne čítať počas tréningu z neho. Po zapísaní príznakov každého obrázku pre všetky veľkosti do súboru bez náhodného horizontálneho otočenia, náš tréning trval približne 33 minút. Dosiahli sme ale požadované rovnaké výsledky ako pri pôvodnom 55 minútovom tréningu, dokonca o trochu lepšie. Podarilo sa nám teda zrýchliť tréning takmer o po-

loviču. V tabuľke 3.5 vidíme výsledky tréningu a na obrázku 3.7 porovnanie s pôvodným tréningom.

Presnosť	Base	Novel	Priemer
mAP	49.184	10.262	38.064
mAP50	77.915	19.747	61.296
mAP75	54.58	9.522	41.707

Tabuľka 3.5: Tabuľka presnosťí na testovacích dátach 2. pokusu o zrýchlenie pri batch size = 1.



Obr. 3.7: Graf porovnania mAP50 na testovacích dátach pôvodného algoritmu a 2. pokusu o zrýchlenie pri batch size = 1.

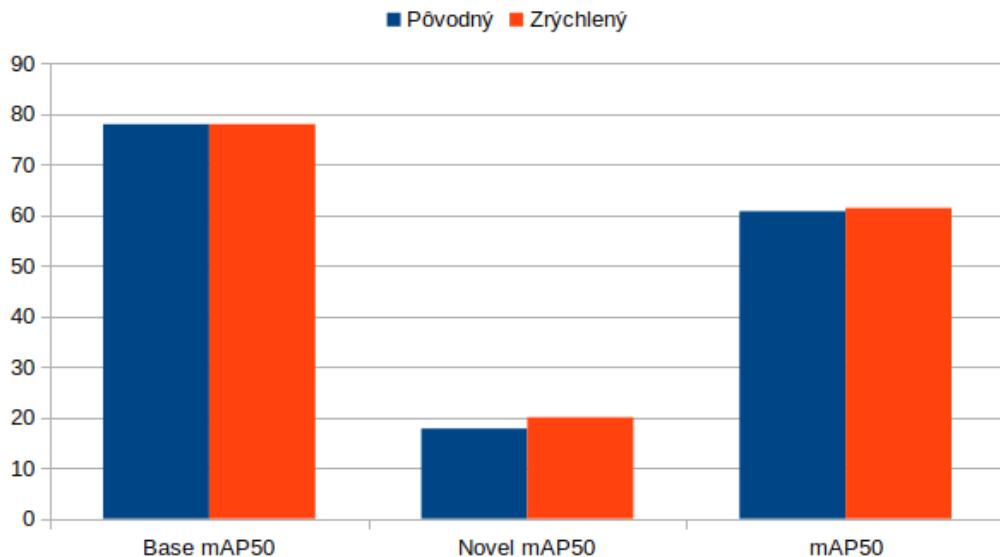
3.2.2 Zapamätanie si výstupu priamo z Box Head

Ako sme videli v predošlej časti, uloženie výstupov z Backbone vrstvy do súborov, pre každý obrázok a pre každú veľkosť zvlášť, nám urýchliло nás tréning takmer o dvojnásobok a zachovala sa naša presnosť. Pokúsime sa teda tréning ešte zrýchliť uložením si výstupov priamo z Box Head a RPN, keďže tieto dva výstupy sú vstupom pre Box Predictor (posledná vrstva

ktorú chceme trénovať). To by malo následne vo zvyšku tréningu výpočtovú zložitosť znížiť a ešte zrýchliť náš tréning. Uvidíme ako veľmi to zrýchli náš tréning v porovnaní z predošlím tréningom.

Presnosť	Base	Novel	Priemer
mAP	49.345	10.218	38.166
mAP50	78.011	20.032	61.446
mAP75	54.54	9.401	41.643

Tabuľka 3.6: Tabuľka presnosťí na testovacích dátach finálneho zrýchleného algoritmu.



Obr. 3.8: Graf porovania mAP50 na testovacích dátach pôvodného a zrýchleného algoritmu.

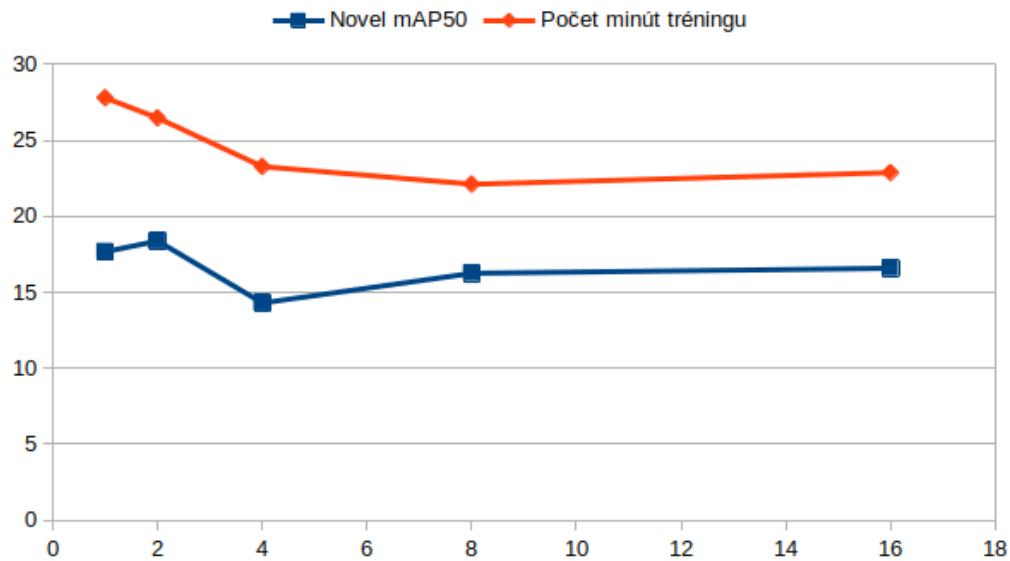
V tabuľke 3.6 vidíme výsledky nášho pokusu a na obrázku 3.8 vidíme porovnanie s pôvodným pomalým algoritmom. Vidíme, že sa nám podarila dosiahnuť rovnaká presnosť. A vyrazne sa nám podarilo zrýchliť tréning celkový čas tréningu bol 6 minút a 51 sekúnd, čo je takmer 8x rýchlejšie ako pôvodný tréning.

Riešenie pre rôzne batch size

Podarilo sa nám náš tréning výrazne zrýchliť, avšak s obmedzením na trénovanie s batch size = 1. Problém s väčším batchom bol ten, že obrázky, ktoré boli spolu v batchi s rôznou veľkosťou a pomerom strán, tak k menšiemu obrázku bol pridaný padding, aby spolu rozmerovo sedeli a keď som si zapamätať príznaky pre ten obrázok s paddingom, keď bol následne v batchi s menším obrázkom a nebol mu pridaný žiadny padding jeho rozmer sa líšil a musel som si zapamätať výstup aj pre tento rozmer. A teda pri väčšej batch size som si musel pamätať výstupy z Box Head a RPN pre každý obrázok v každom jeho rozmere v ktorom sa objavil v batchi a teda oveľa viac dát, čo spôsobovalo veľmi vysokú pamäťovú náročnosť. Tento problém už však nemáme, keďže výstupy z vrstiev si ukladáme do súborov na disk.

Po upravení implementácie aby fungoval náš algoritmus pre rôzne batch size, sme schopný trénovať na rôznych batch size na našom GPU, dokonca aj vyššie hodnoty ako napríklad 16, keďže náš algoritmus ukladaním výstupov do textových súborov dosahuje oveľa nižšiu pamäťovú náročnosť pre GPU. Miesto toho aby si pamätať výstupy z jednotlivých vrstiev pre celý batch počas jednej iterácie, si ich nemusí pamätať a načíta si ich zo súboru. A pri ukladaní výstupov to robí postupne pre každý obrázok zvlášť.

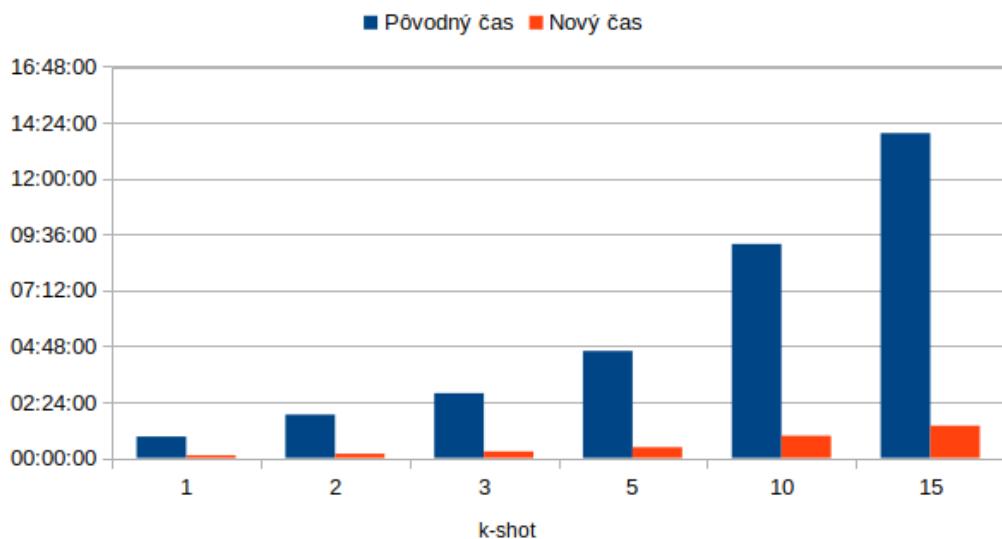
Vyskúšame porovnať presnosť a dĺžku 1-shot tréningu pri rôznych batch size s tým, že prispôsobíme počet iterácií a learning rate. Pri zdvojnásobení batch size, zmenšíme počet iterácií na polovicu a zdvojnásobíme learning rate.



Obr. 3.9: Porovnanie presnosti na testovacích dátach a dĺžky tréningu rôznych batch size.

Na obrázku 3.9 vidíme porovnanie novel mAP50 a dĺžky tréningu pri rôznych batch size. Vidíme, že čas tréningu je veľmi podobný a najvyššiu presnosť dosahujeme pri batch size = 2. Pri tomto pokuse sme použili iné náhodné obrázky na tréning, takže presnosť novel mAP50 sa trochu líši od predošlých pokusov.

Vyhodnotenie



Obr. 3.10: Porovnanie rýchlosí pôvodného algoritmu a algoritmu po zrýchlení

Na obrázku 3.10 vidíme porovnanie rýchlosi pôvodného algoritmu a algoritmu po zrýchlení pri rôznych k-shot tréningoch. Oba algoritmy dosahujú podobnú presnosť.

Kapitola 4

Testovanie algoritmu pri zmene počtu tried

V predošlých kapitolách sme si ukázali ako sa mení presnosť a rýchlosť tréningu pri zmene počtu trénovacích obrázkov pre 6 novel tried a taktiež sa nám podarilo poslednú fázu tréningu(fine tuning) 8 násobne zrýchliť. Teraz využijeme náš zrýchlený algoritmus a pozrieme sa na to ako sa mení presnosť a rýchlosť pri zvyšovaní počtu novel tried.

4.1 Fine-tuning na base + novel triedach

Najprv vyskúšame ako sa nám mení presnosť pri zvyšovaní počtu novel tried, pri ponechaní detekcie 15 base tried, rovnako ako doteraz. Neskôr si vyskúšame spraviť fine-tuning len na novel triedach, čo by malo zvýšiť našu presnosť pre novel triedy, s tým, že base triedy nebudeme detegovať.

Pre jednoduchšie a rýchlejšie pridávanie nových tried som napísal zopár scriptov. Popíšem postup ako pridať novú triedu do datasetu:

Anotované obrázky pre nové triedy stiahneme z roboflow [3] vo formate

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED56

Pascal VOC XML. Pre pridanie novej triedy do datasetu treba rozdeliť anotované obrázky na trénovacie a testovacie. Stačí keď presunieme všetky anotované obrázky do priečinka `datasets/VOC2007/ImageSets/Main/test`. Následne spustíme script `datasets/VOC2007/ImageSets/Main/divideDataset.py`, ktorý rozdelí anotované obrázky do priečinkov `train` a `test`. Keďže na fine-tuning nebudeme potrebovať príliš trénovacích obrázkov tak do priečinka `train` presunieme 30 obrázkov a zvyšok necháme v priečinku `test`. Potom treba spustiť script `datasets/VOC2007/ImageSets/Main/addNewClass.py`, ktorý pridá nové obrázky do textových súborov `trainval.txt` a `test.txt` a presunie anotácie do `datasets/VOC2007/Annotations` a obrázky do `datasets/VOC2007/JPEGImages`.

Potom treba spustiť script `datasets/prepare_voc_few_shot.py`, ktorý vytvorí textové súbory pre jednotlivé k-shot tréningy pre každú triedu. Vo formáte `box_kshot_názov_triedy_train.txt`, každý z týchto súborov obsahuje k názvov náhodných obrázkov z danej triedy, ktoré berie z textového suboru `trainval.txt` a následne sa trénuje a validuje počas fine-tuningu len na nich. Všetky obrázky v testovacej množine sú využité pri testovaní, na vyhodnotenie nášho modelu.

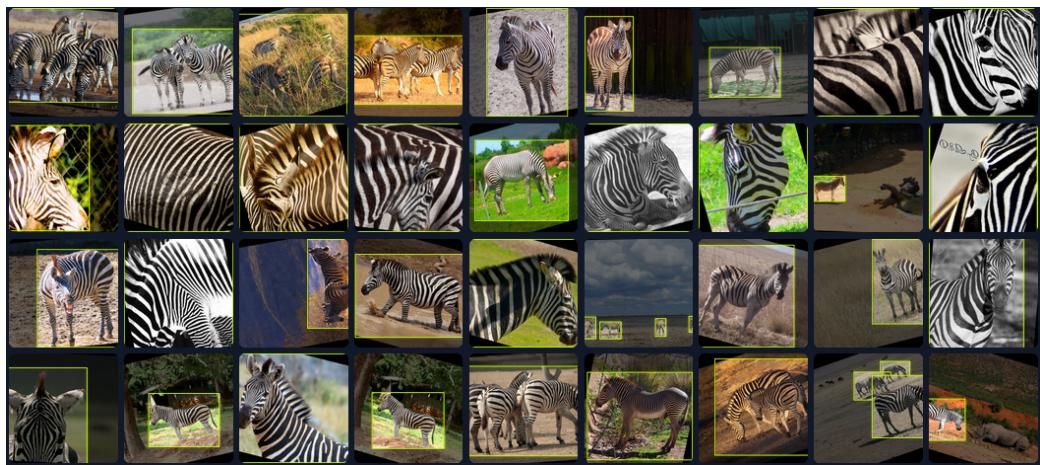
Rozhodol som sa používať pri tomto testovaní 10-shot detekciu, pretože dosahuje vysokú presnosť, rozptyl medzi presnosťou jednotlivých tried je nízky a vieme ju vykonať po našom zrýchlení pomerne rýchlo za cca hodinu.

Nové triedy som vyberal úplne náhodne a keďže závisí presnosť detekcie na ich podobnosti s base triedami, ukážeme si taktiež AP50 pre jednotlivé novel triedy.

10 novel tried

Na obrázkoch 4.1 až 4.4 vidíme príklady anotovaných obrázkov pre novo pridané novel triedy.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED 57

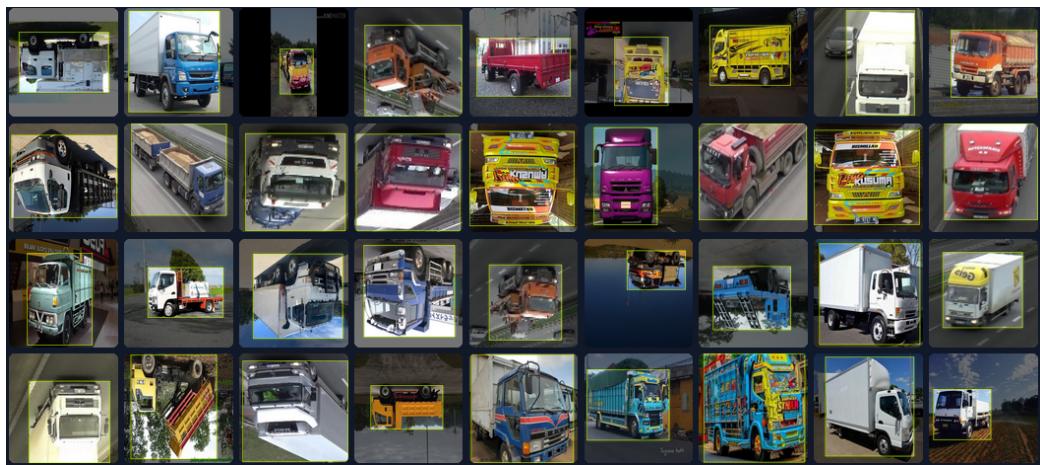


Obr. 4.1: Príklady anotovaných obrázkov pre triedu zebra

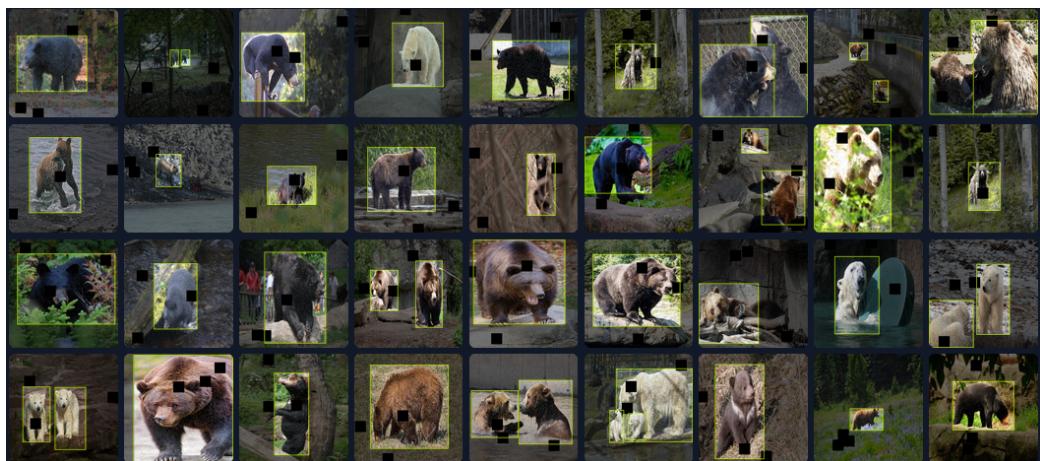


Obr. 4.2: Príklady anotovaných obrázkov pre triedu pomaranč

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED58



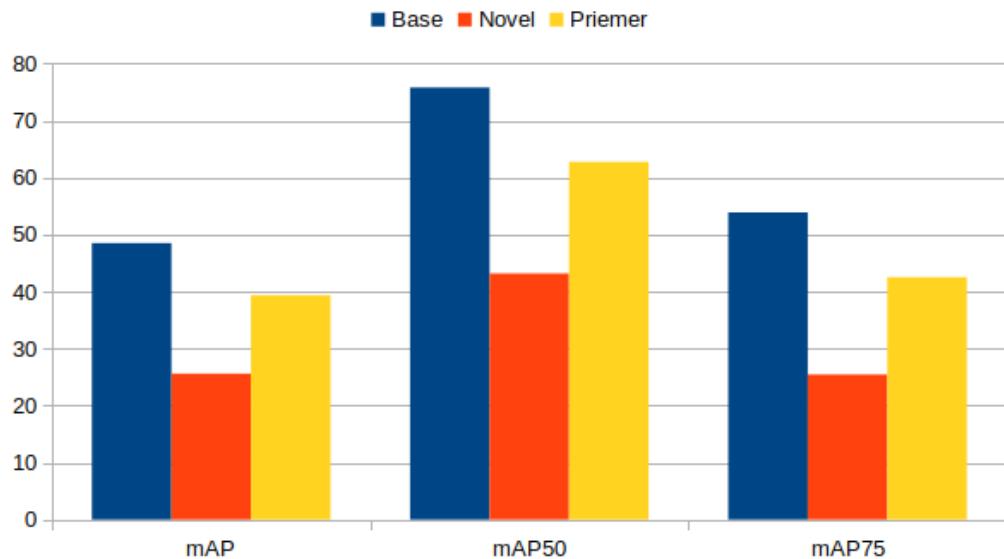
Obr. 4.3: Príklady anotovaných obrázkov pre triedu nákladné vozidlo



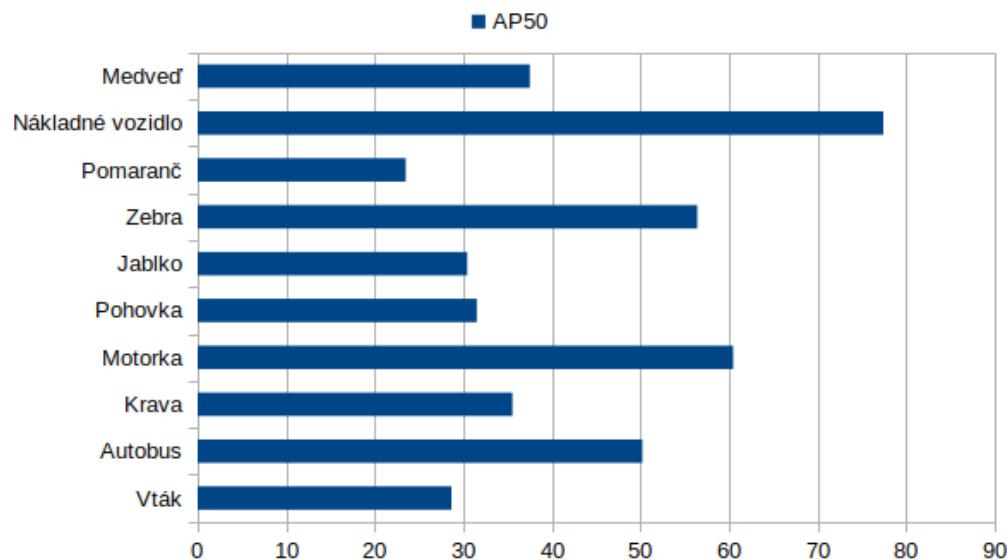
Obr. 4.4: Príklady anotovaných obrázkov pre triedu medveď

Tréning trval 1 hodinu, 7 minút a 51 sekúnd. Dĺžka tréningu sa nám o trochu zvýšila oproti 6 novel triedam a taktiež nám trochu klesla presnosť. Ako vidíme na obrázku 4.6 presnosti sú pomerne dobre rozložené a nové triedy si držia solidnú presnosť.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED59



Obr. 4.5: Graf presnosti na testovacích dátach pre 10 novel tried + 15 base tried pri 10-shot detekcii.



Obr. 4.6: AP50 na testovacích dátach pre 10 novel tried pri 10-shot detekcii

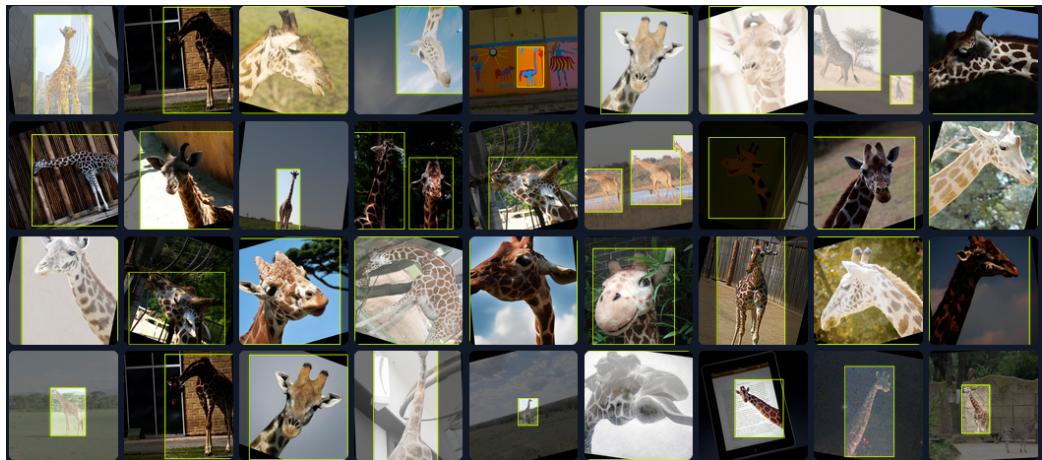
KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED60

Presnosť	Base	Novel	Priemer
mAP	48.508	25.594	39.342
mAP50	75.803	43.18	62.754
mAP75	53.901	25.442	42.517

Tabuľka 4.1: Tabuľka presnosťí na testovacích dátach pre 10 novel tried + 15 base tried pri 10-shot detekcii.

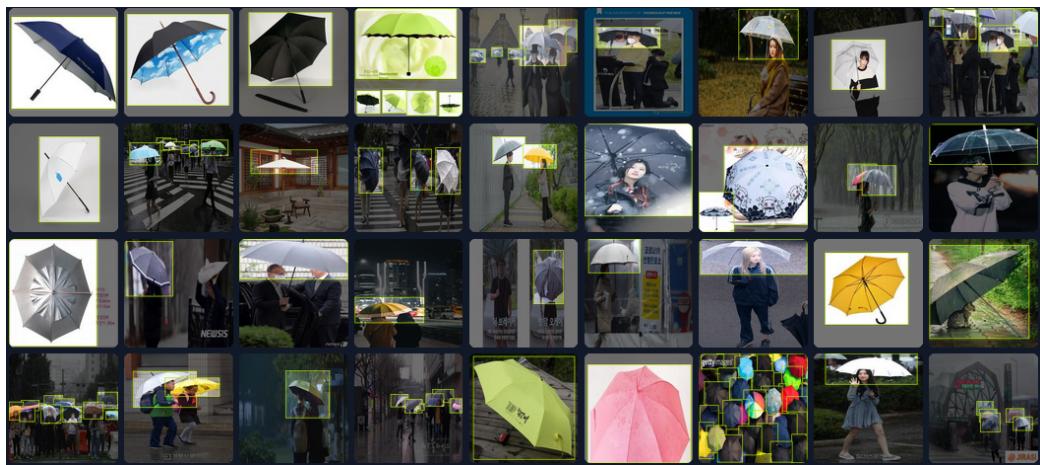
15 novel tried

Na obrázkoch 4.7 - 4.11 vidíme príklady anotovaných obrázkov pre novo pridané novel triedy.

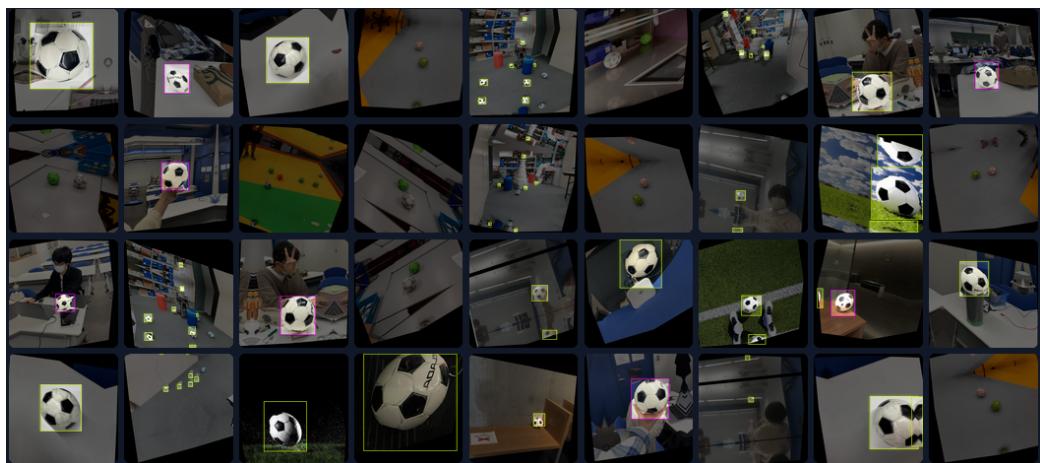


Obr. 4.7: Príklady anotovaných obrázkov pre triedu žirafa

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED61

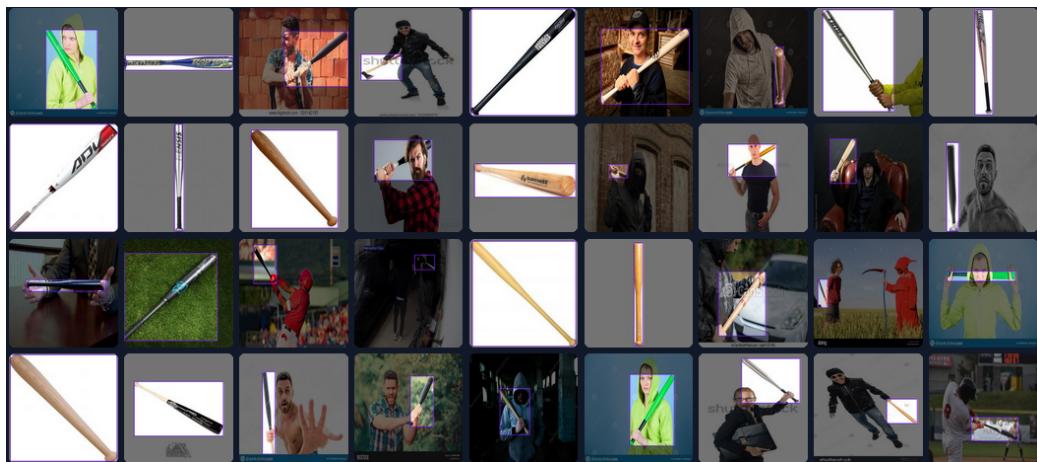


Obr. 4.8: Príklady anotovaných obrázkov pre triedu dážnik

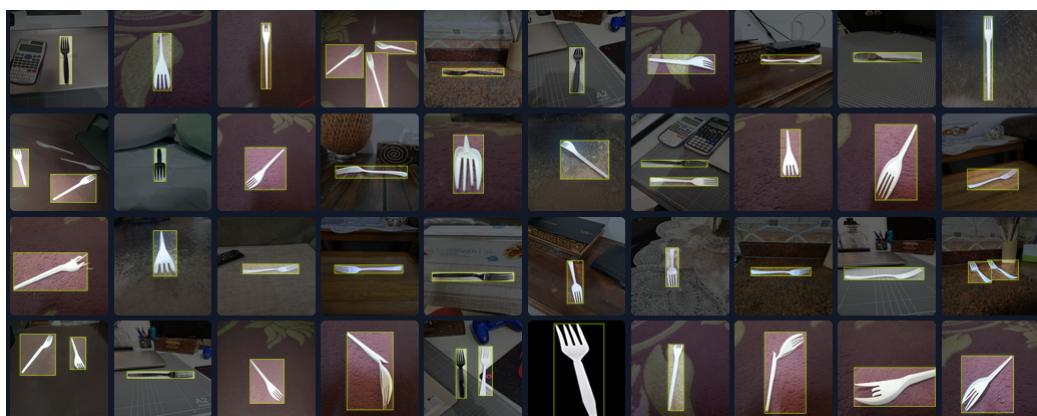


Obr. 4.9: Príklady anotovaných obrázkov pre triedu futbalová lopta

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED62



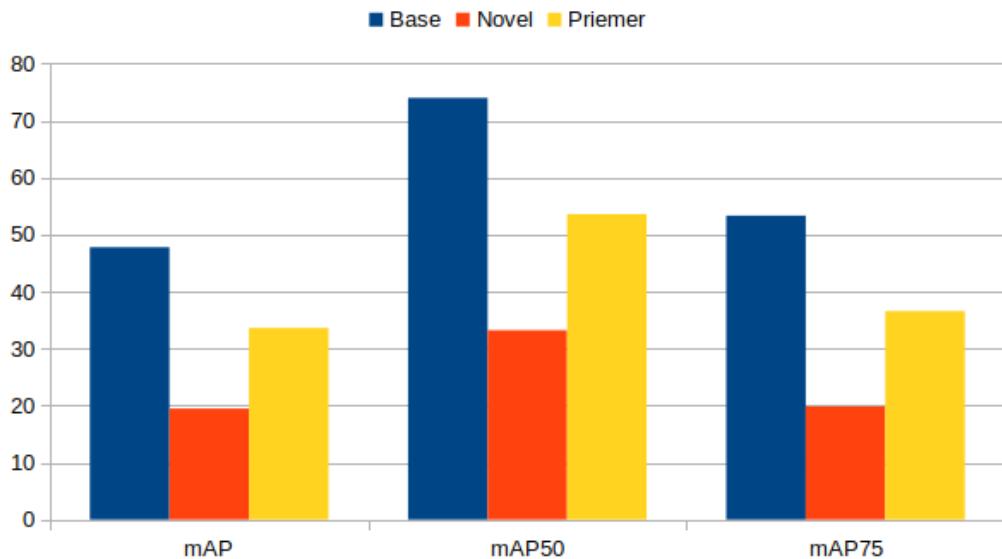
Obr. 4.10: Príklady anotovaných obrázkov pre triedu basebalová pálka



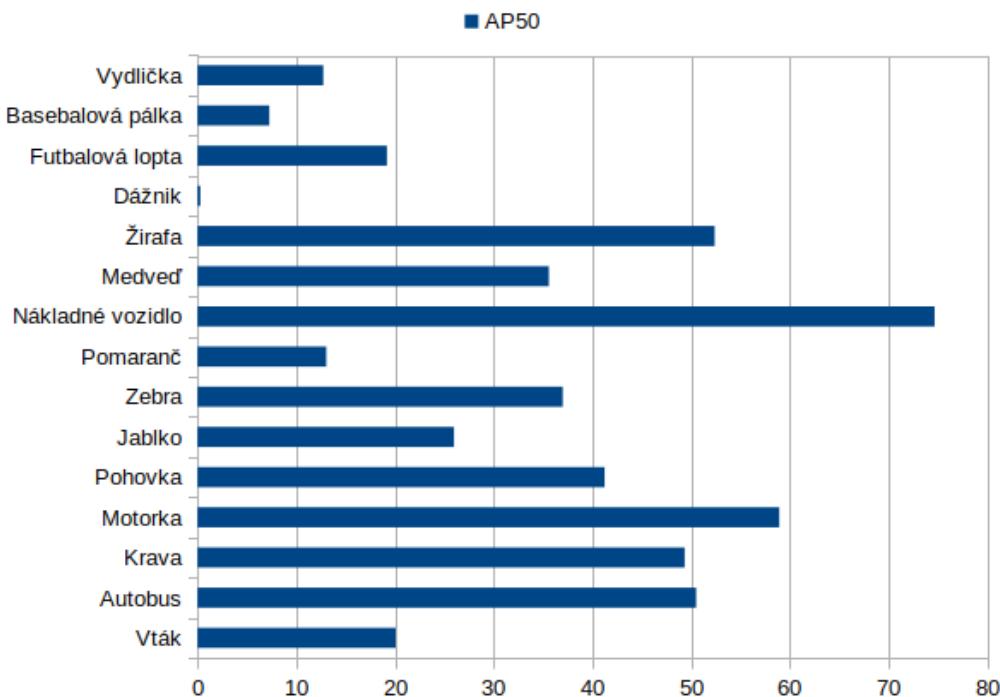
Obr. 4.11: Príklady anotovaných obrázkov pre triedu vydlička

Tréning trval 1 hodinu, 18 minút a 41 sekúnd. V tabuľke 4.2 a na obrázku 4.12 vidíme presnosť pri 15 novel triedach. Vidíme, že presnosť nám klesla a na obrázku 4.13 vidíme, že sa nám taktiež zvýšila odchýlka v presnosti medzi triadami a niektoré triedy majú dosť nízku presnosť, trieda dážnik má takmer nulovú.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED63



Obr. 4.12: Graf presnosti na testovacích dátach pre 15 novel tried + 15 base tried pri 10-shot detekcii.



Obr. 4.13: AP50 na testovacích dátach pre 15 novel tried pri 10-shot detekcii.

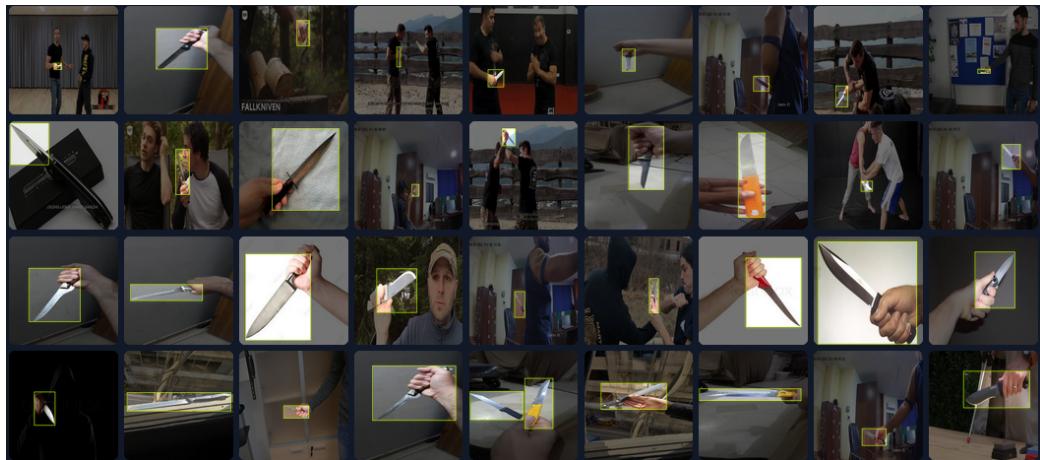
KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED64

Presnosť	Base	Novel	Priemer
mAP	47.779	19.471	33.625
mAP50	73.987	33.208	53.597
mAP75	53.33	19.824	36.577

Tabuľka 4.2: Tabuľka presnosťí na testovacích dátach pre 15 novel tried + 15 base tried pri 10-shot detekcii.

20 novel tried

Na obrázkoch 4.14 - 4.18 vidíme príklady anotovaných obrázkov pre novo pridané novel triedy.



Obr. 4.14: Príklady anotovaných obrázkov pre triedu nôž

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED 65



Obr. 4.15: Príklady anotovaných obrázkov pre triedu banán

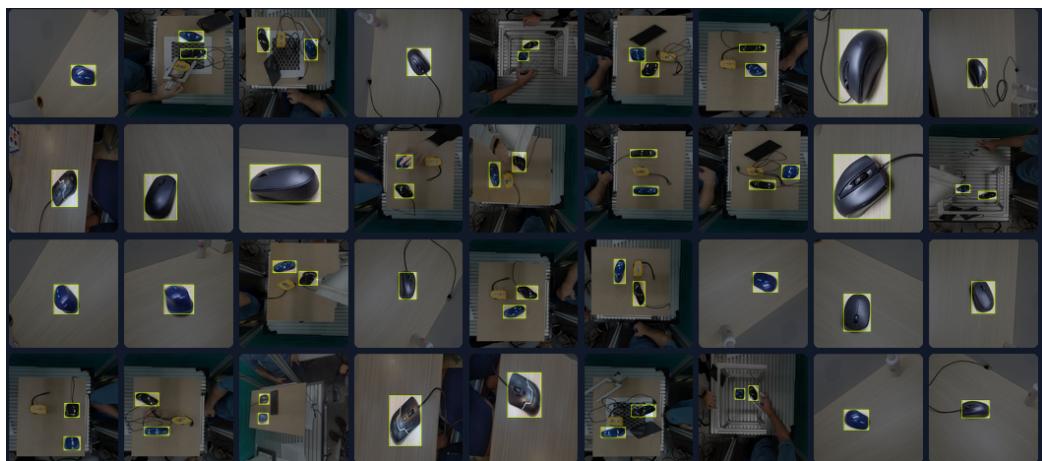


Obr. 4.16: Príklady anotovaných obrázkov pre triedu mrkva

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED66



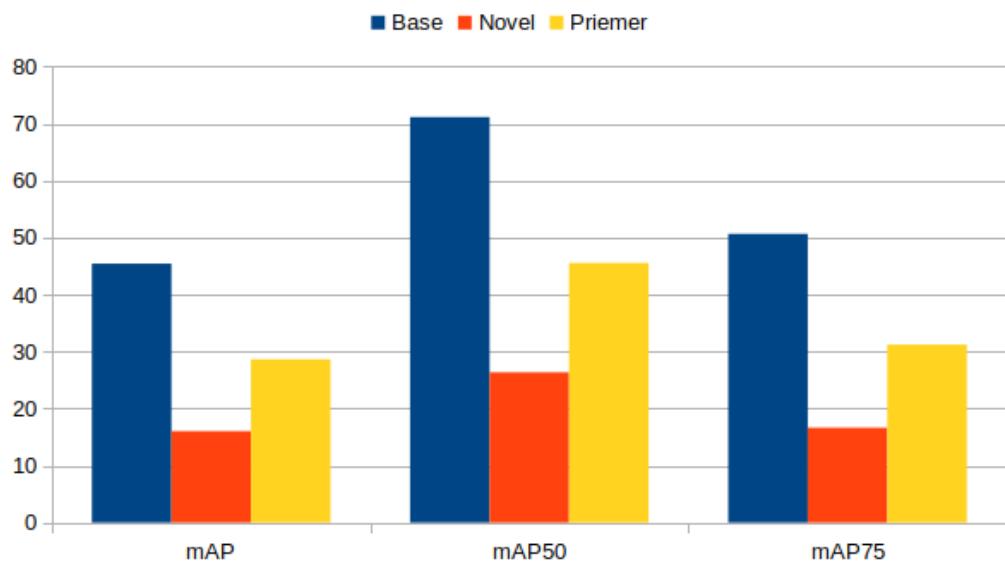
Obr. 4.17: Príklady anotovaných obrázkov pre triedu hotdog



Obr. 4.18: Príklady anotovaných obrázkov pre triedu počítačová myš

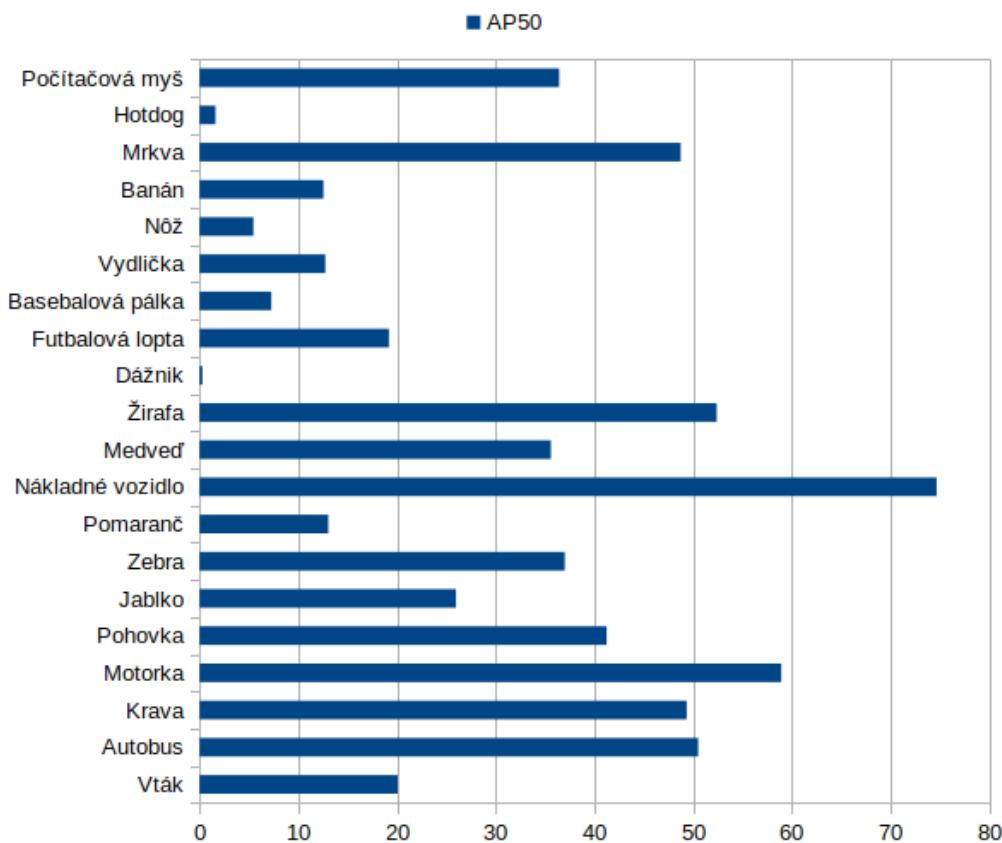
Tréning trval 1 hodinu, 19 minút a 23 sekúnd. V tabuľke 4.3 a na obrázku 4.19 vidíme presnosť pri 20 novel triedach. Na obrázku 4.20 vidíme presnosť jednotlivých novel tried. Nové triedy Hotdog, Nôž, Dážnik a Basebalová pálka majú veľmi nízku presnosť AP50 pod 10.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED67



Obr. 4.19: Graf presnosti na testovacích dátach pre 20 novel tried + 15 base tried pri 10-shot detekcii.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED68



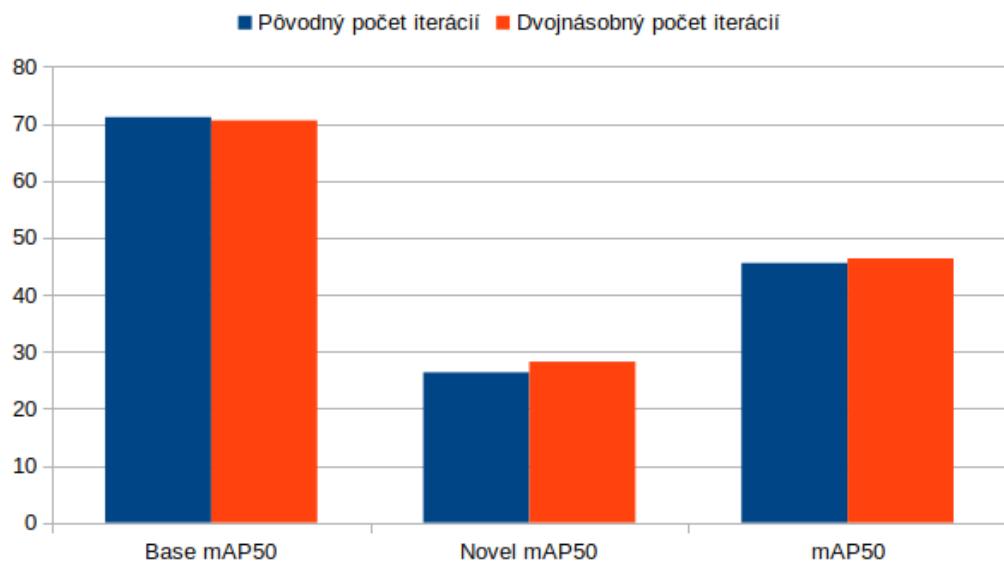
Obr. 4.20: AP50 na testovacích dátach pre 20 novel tried pri 10-shot detekcii.

Presnosť	Base	Novel	Priemer
mAP	45.432	16.031	28.631
mAP50	71.154	26.336	45.544
mAP75	50.659	16.635	31.217

Tabuľka 4.3: Tabuľka presnosťí na testovacích dátach pre 20 novel tried + 15 base tried pri 10-shot detekcii.

Vidíme, že pri zvyšovaní počtu tried nám presnosť nášho tréningu stále klesá. Vyskúšame zvýšiť počet iterácií tréningu, keďže počet trénovacích obrázkov sa nám zvýšil, skúsime zvýšiť počet iterácií pri 20 novel triedach na dvojnásobok, teda aj čas tréningu bude dvojnásobne dlhý a porovnáme ich presnosť.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED69

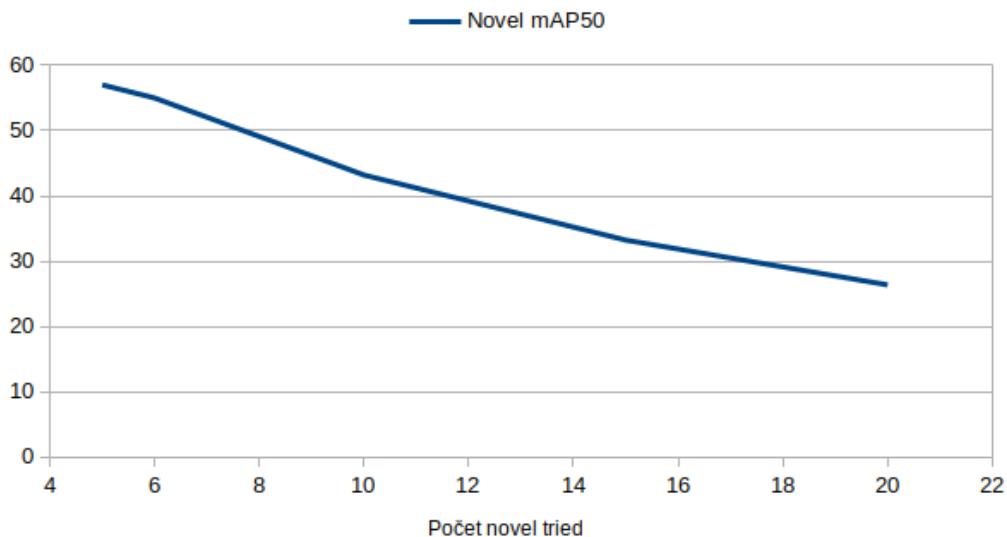


Obr. 4.21: Porovnanie presnosti na testovacích dátach 10-shot detekcie s 20 novel triedami + 15 base triedami s pôvodným a dvojnásobným počtom iterácií

Ako vidíme na obrázku 4.21, pri zdvojnásobení počtu iterácií sa nám presnosť trochu zvýšila avšak len minimálne na úkor zdvojnásobenia dĺžky tréningu, takže to nebolo príliš efektívne.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED70

Vyhodnotenie



Obr. 4.22: Presnosť 10-shot detekcie na testovacích dátach vzhľadom k počtu novel tried

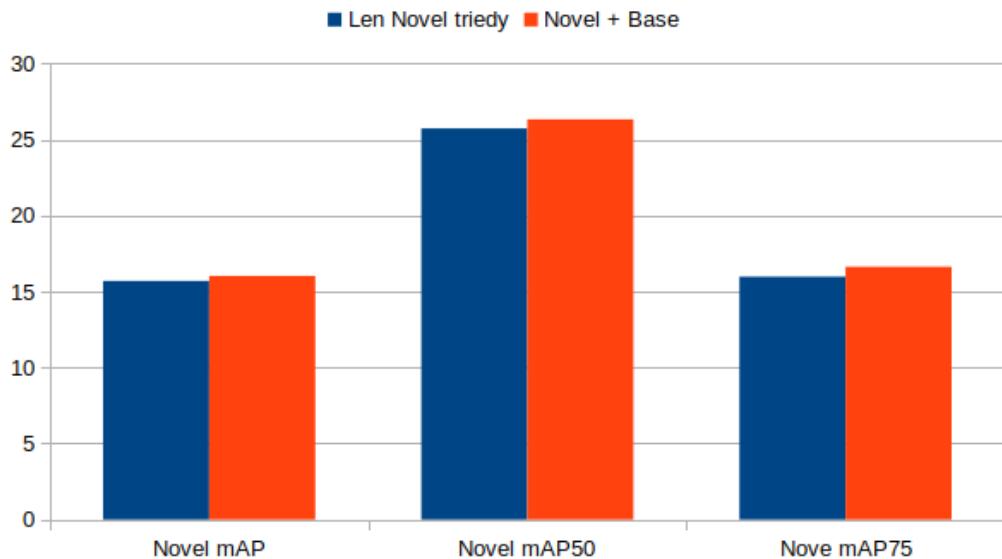
Ako vidíme na obrázku 4.22 presnosť nám pri zvyšovaní počtu novel tried konštantne klesá. Ak by sme chceli udržať našu presnosť pri viacej novel triedach ideálne by bolo natrénovať viacej modelov každý pre detekciu iných tried.

4.2 Fine-tuning len na novel triedach

Ak nepotrebujeme detegovať base triedy, môžu byť využité len pri 1.fáze tréningu (base tréning), na predtrénovanie celej siete. Následný fine-tuning môžme robiť len na novel triedach, čo by malo zvýšiť presnosť ich detekcie.

Vyskúšame spraviť 10-shot fine-tuning na 20 novel triedach, bez base tried a porovnáme presnosť detekcie pri detekcii 20 novel tried spolu s 15 base triedami.

KAPITOLA 4. TESTOVANIE ALGORITMU PRI ZMENE POČTU TRIED71



Obr. 4.23: Graf porovnania presnosti na testovacích dátach pre fine-tuning bez a s base triedami pri 10-shot detekcii

Presnosť	Len Novel triedy	Novel + Base
Novel mAP	15.696	16.031
Novel mAP50	25.74	26.336
Novel mAP75	15.978	16.635

Tabuľka 4.4: Tabuľka porovnania presnosti na testovacích dátach pre fine-tuning bez a s base triedami pri 10-shot detekcii

Ako vidíme v tabuľke 4.4 a na obrázku 4.23 fine-tuning čisto na novel triedach nenaplnil naše očakávania a presnosť detekcie novel tried sa dokonca trochu znížila.

Kapitola 5

Testovanie tréningu modelu pre detekciu objektov pomocou kamery robota NICO

V tejto kapitole vyskúšame natrénovať model pre detekciu mandarinky na záberoch z kamery robota NICO pomocou nášho zrýchľeného algoritmu.

5.1 Testovanie na pôvodných snímkoch z kamery

Nafotili sme približne 150 obrázkov obsahujúcich mandarinku pomocou kamery levého oka robota NICO. Nafotené obrázky boli však rôznej kvality a veľa z nich bolo veľmi nekvalitných a zašumených. Tak sme vybrali 40 najlepších záberov na tréning a vyhodnotenie modelu. Obrázky anotujeme pomocou knižnice `labelImg` [19].

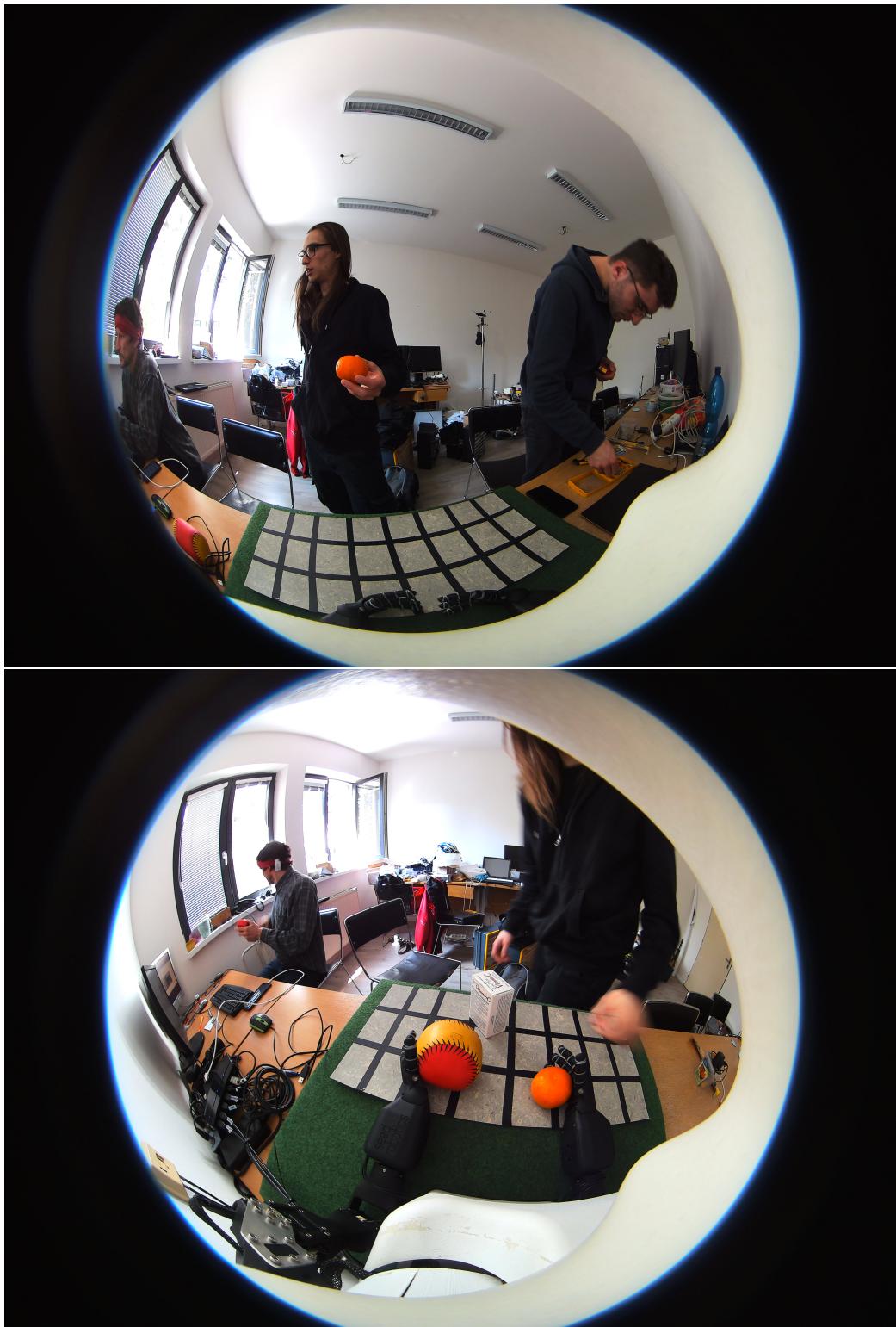
Model budeme trénovať pomocou 10-shot fine tuningu. Použijeme 10

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC

obrázkov na tréning a zvyšných 30 na testovanie modelu. Náš model budeme trénovať iba na jednej triede pre maximalizáciu presnosti. Skúšal som rôzne parametre pre learning rate, batch size a počet iterácií, no najvyššiu presnosť som dosiahol pre batch size = 8, learning rate = 0.0005, počet iterácií = 8 000 a step = 7 200 (počet iterácií po ktorých sa learning rate desaťnásobne zníži).

Na obrázku 5.1 vidíme príklady trénovacích obrázkov, na obrázku 5.2 vidíme príklady úspešných detekcií na testovacích obrázkoch, na obrázku 5.3 vidíme príklady neúspešných detekcií na testovacích obrázkoch, v tabuľke 5.1 vidíme výslednú presnosť nášho modelu a v tabuľke 5.2 vidíme jadnoduchšiu presnosť pre lepšiu interpretáciu, pri confidence threshold = 0.5. Pri znížení confidence threshold sme nedostali viac správnych detekcií len sa nám zvyšoval počet nesprávnych.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC



Obr. 5.1: Príklad trénovacích obrázkov z kamery robota NICO.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC



Obr. 5.2: Príklad úspešných detekcií testovacích obrázkov z kamery robota NICO.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC



Obr. 5.3: Príklad neúspešných detekcií testovacích obrázkov z kamery robota NICO.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOCOU SNÍMOK Z KAMERY

Trieda	mAP	AP50	AP75
Mandarinka	14.545	18.182	18.182

Tabuľka 5.1: Tabuľka presnosti na testovacích dátach pri detekcií mandarinky pomocou snímok z kamery robota NICO.

Trieda	Počet obrázkov	Úspešné	Neúspešné	Nedetegované
Mandarinka	30	4	0	26

Tabuľka 5.2: Tabuľka jednoduchšej presnosti na testovacích dátach pri detekcií mandarinky pomocou snímok z kamery robota NICO pri confidence threshold = 0.5.

5.2 Testovanie na snímkoch z kamery zbavených skreslenia

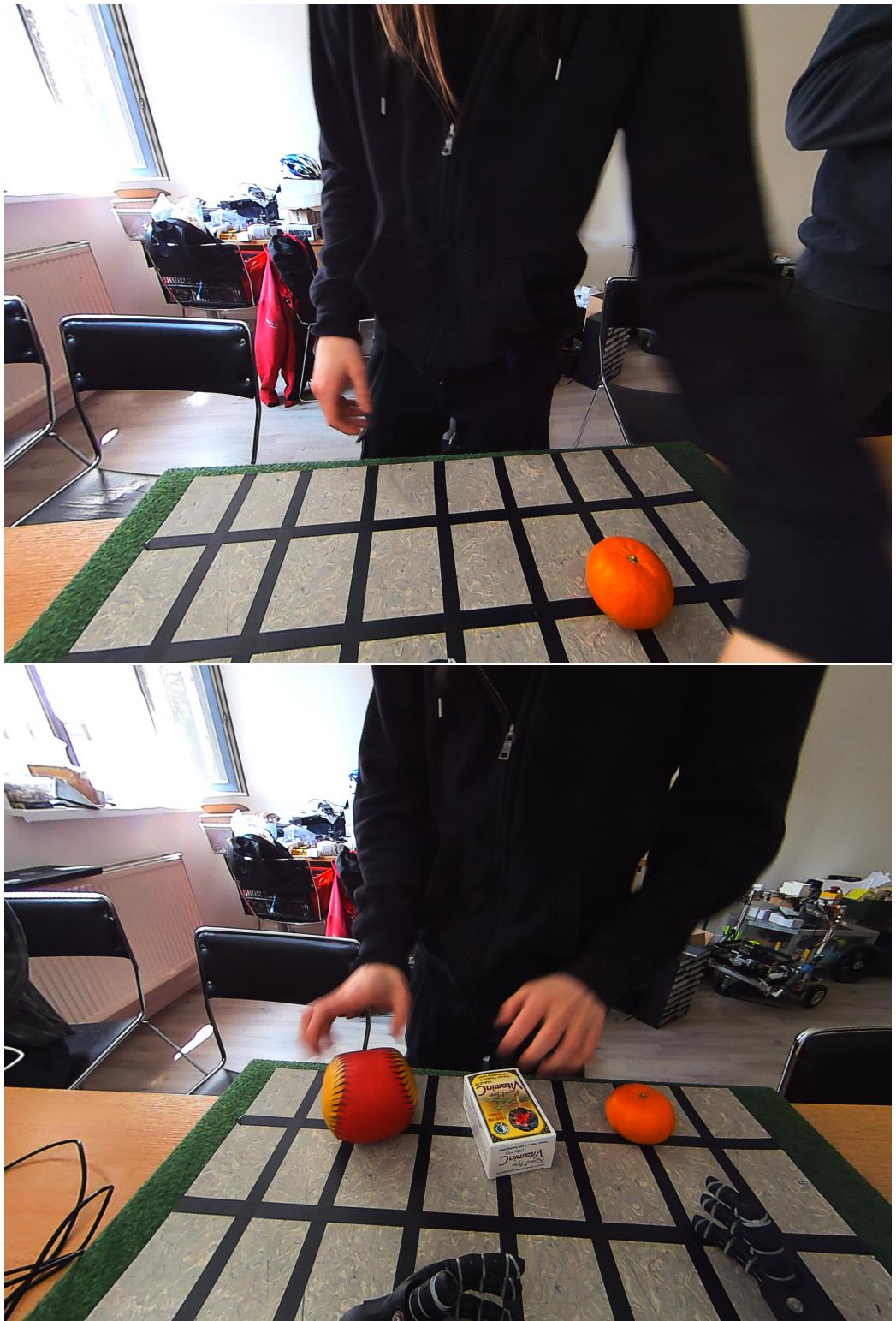
Ako vidíme naša presnosť, nie je príliš vysoká, za čo zrejmä môže aj nižšia kvalita našich obrázkov. Teraz vyskúšame natrénovať a vyhodnotiť model na obrázkoch bez skreslenia. Na zbavenia sa skreslenia použijeme knižnicu `planar_nico_vision` [20]. Po odstránení skreslenia, sme na niektorých obrázkoch odrezali taktiež mandarinku. Takže sa náš dataset kvalitných obrázkov zredukoval na 30. Rozdelíme teda dataset na 10 trénovacích a 20 testovacích obrázkov.

Na obrázku 5.4 vidíme príklady trénovacích obrázkov, na obrázku 5.5 vidíme príklady úspešných detekcií na testovacích obrázkoch, na obrázku 5.6 vidíme príklady neúspešných detekcií na testovacích obrázkoch, v tabuľke 5.3 vidíme výslednú presnosť nášho modelu a v tabuľke 5.4 vidíme jednoduchšiu presnosť nášho modelu pre lepšiu interpretáciu, pri confidence threshold = 0.1. Naša presnosť bola taká nízka, že sme museli zísiť confidence threshold aby sme dostali aspoň nejaké správne detekcie.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC

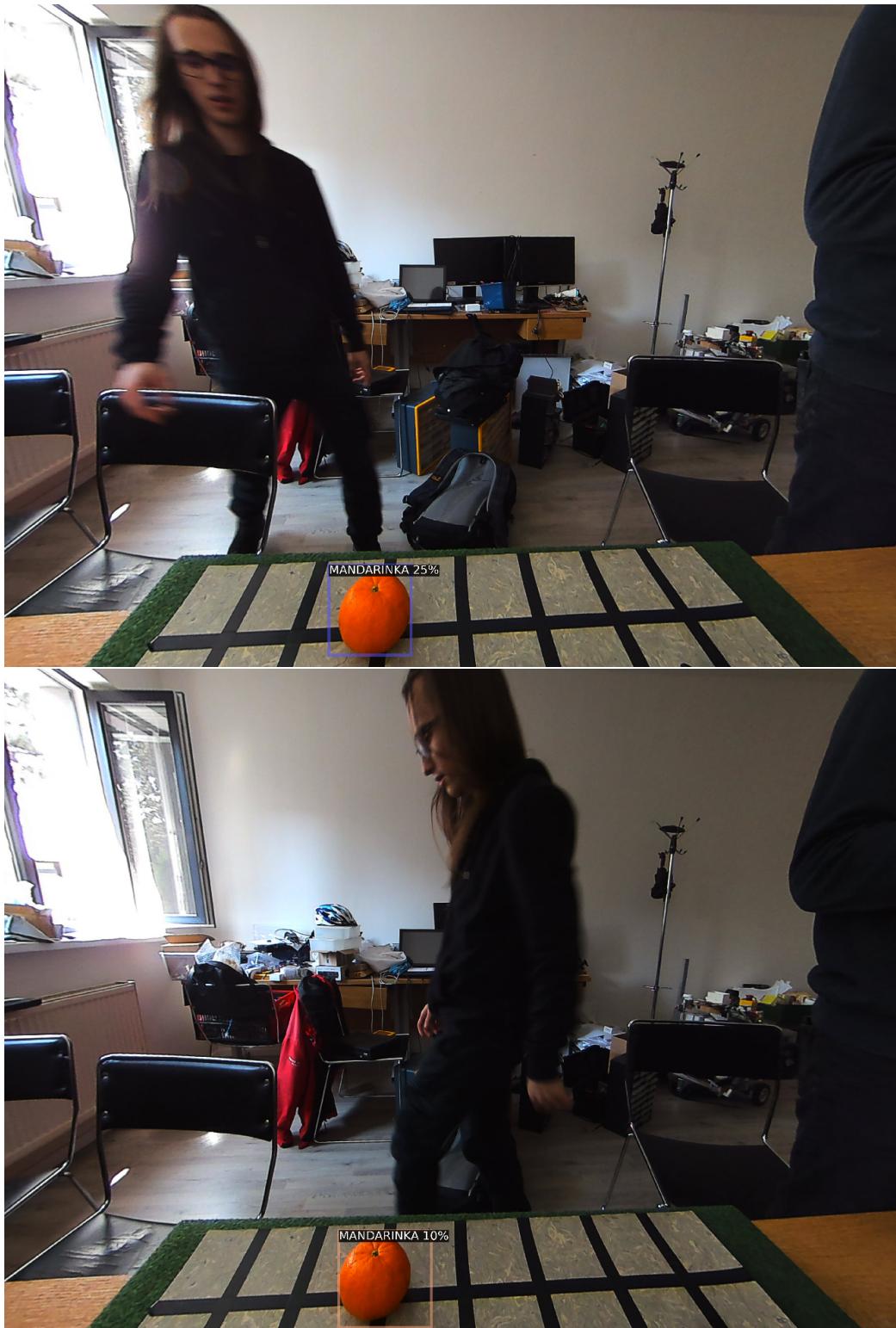
Vidíme, že zbavenie skreslenia nám nepomohlo a dosahujeme ešte nižšiu presnosť ako z pôvodných, neupravených snímkov.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC



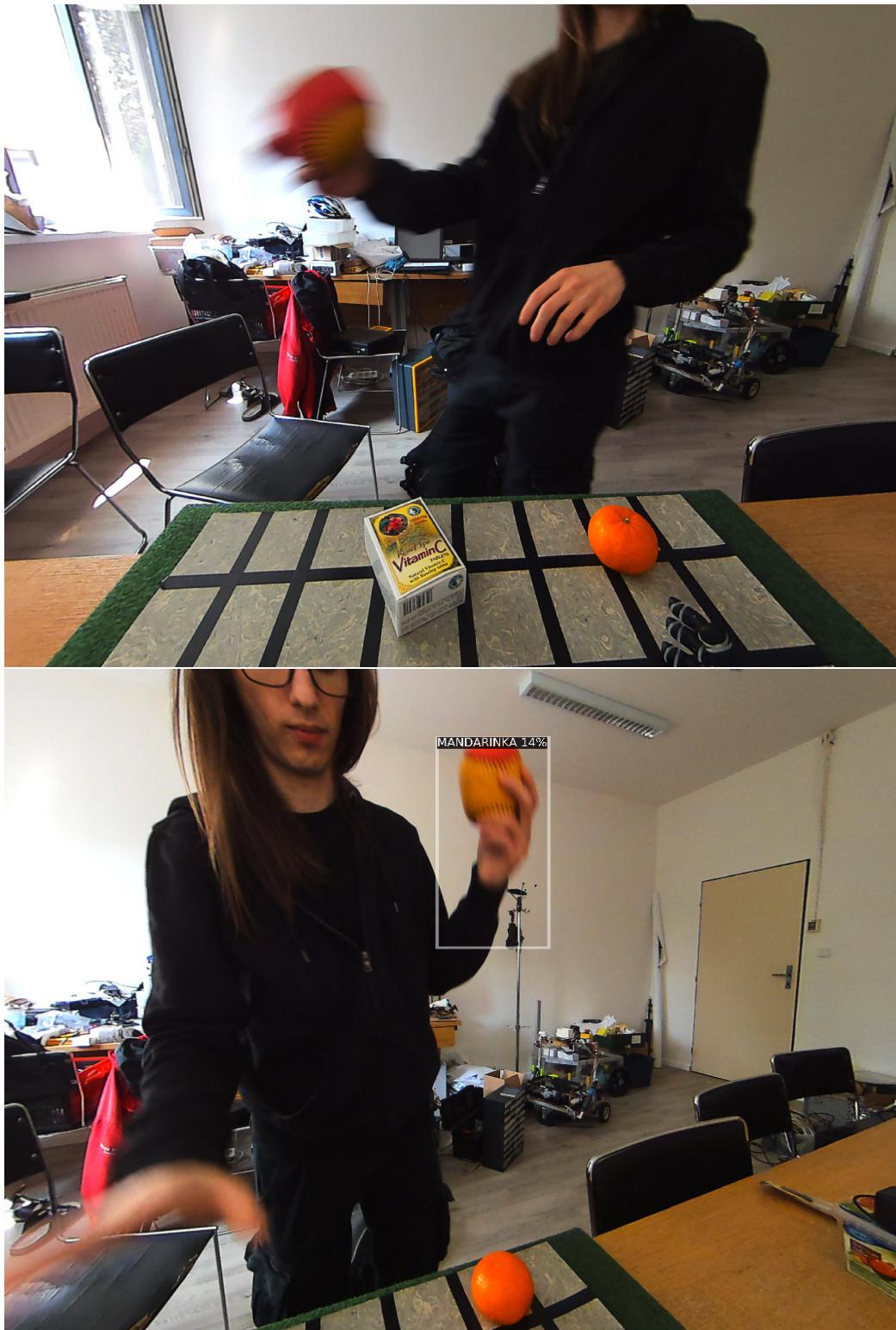
Obr. 5.4: Príklad trénovacích obrázkov z kamery robota NIC0 zbavených skreslenia.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC



Obr. 5.5: Príklad úspešných detekcií testovacích obrázkov z kamery robota NICO zbavených skreslenia.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC



Obr. 5.6: Príklad neúspešných detekcií testovacích obrázkov z kamery robota NICO zbavených skreslenia.

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOCOU SNÍMOK Z KAMERY ROBOTA NICO

Trieda	mAP	AP50	AP75
Mandarinka	2.545	4.848	3.030

Tabuľka 5.3: Tabuľka presnosti na testovacích dátach pri detekcií mandarinky pomocou snímok z kamery robota NICO zbavených skreslenia.

Trieda	Počet obrázkov	Úspešné	Neúspešné	Nedetegované
Mandarinka	20	2	7	18

Tabuľka 5.4: Tabuľka jednoduchšej presnosti na testovacích dátach pri detekcií mandarinky pomocou snímok z kamery robota NICO zbavených skreslenia.

Discussion

V kapitole 2 sme otestovali rýchlosť a presnosť pôvodného algoritmu. Z týchto hodnôt sme následne vychádzali pri porovnaní s našim vylepšením tohto algoritmu. Pri 1-shot detekcii sme dosiahli presnosť novel mAP = 9.41, tréning trval 55 min. a tréning sme mohli vykonávať maximálne pre batch size = 2, kvôli pamäťovej náročnosti na GPU. Zistili sme taktiež, že čas tréningu nám lineárne rastie pri zvyšovaní počtu trénovacích obrázkov, avšak nárast presnosti nie je lineárny a medzi 10-shot a 15-shot detekciou je minimálny rozdiel v presnosti.

V kapitole 3 sme robili všetky pokusy pri 1-shot detekcii. Pokúsili sme sa tento algoritmus zrýchliť najprv pomocou zníženia počtu iterácií tréningu, čo bolo neúspešné a mali sme signifikantné zníženie presnosti. Následne sme sa pokúsili pridať padding pre všetky obrázky tak aby mali rozmery najväčšieho obrázku, aby sa po pridaní do batchu nemenil ich rozmer a zapamätať si výstup z backbone pre každý obrázok, aby nám stačilo prejsť backbonom len raz, avšak obrázky boli príliš veľké a nestačila nám na to grafická karta. Potom sme vyskúšali resizovať obrázky na rovnakú veľkosť to však nebolo príliš úspešné a naša presnosť bola nízka v porovnaní s pôvodným algoritmom. Následne sme vyskúšali použiť batch size = 1 a vynechať augmentácie a tak sme si zapamätali výstup z backbone pre každý obrázok, dĺžku tréningu sa nám sice podarilo znížiť na 20 minút avšak presnosť taktiež veľmi klesla

KAPITOLA 5. TESTOVANIE TRÉNINGU MODELU PRE DETEKCIU OBJEKTOV POMOC

na novel mAP = 1.52. Tak sme si skúsili zapamätať príznaky pre každý obrázok vo všetkých jeho rozmeroch, uložením do súboru. Tento pokus bol veľmi úspešný a dosiahli sme presnosť novel mAP = 10.262, pri čase tréningu 33 min. Následne sme tréning ešte zrýchlili tým, že sme vynechali aj ďalšie zamrazené vrstvy a uložili sme si výstup priamo z box head, náš tréning trval iba necelých 7 min. a naša presnosť sa zachovala novel mAP = 10.218. Následne sme upravili implementáciu aby fungovala pre rôznu batch size, keďže sem ukladali výstupy do súborov na disk nezaťažovali pamäť našej GPU. Vyskúšali otestovať presnosť a rýchlosť nášho algoritmu pri rôznych batch size. Najlepšie výsledky dosiahol pri batch size = 2.

V kapitole 4 sme otestovali náš algoritmus pri rôznom počte novel tried, pri zvyšovaní počtu tried sa znižovala aj presnosť. Vyskúšali sme taktiež trénovať iba na novel triedach, avšak nemalo to očakávaný efekt a naša presnosť sa nezvýšila.

V kapitole 5 sme vyskúšali testovať náš algoritmus na snímkoch z kamery robota NICO. Dosiahli sme presnosť mAP = 14.545 pri 10-shot detekcii mandarinky. Pokúsili sme sa zo snímkov odstrániť skreslenie, to však zredukovalo aj počet našich snímkov na ktorých bola mandarinka a naša presnosť sa znížila na mAP = 2.545. Naša presnosť na snímkoch z kamery robota NICO bola oveľa nižšia ako presnosť v predošlých testoch. Myslím, že za to môže hlavne kvalita a nízky počet snímkov pre testovanie.

Záver

V práci sme sa zamerali na few-shot objektovú detekciu, otestovali sme presnosť a rýchlosť algoritmu Frustratingly Simple Few-Shot Object Detection [1] pre rôzny počet trénovacích obrázkov. Následne sa nám podarilo tento algoritmus zrýchliť, znížiť pamäťovú náročnosť a udržať rovnakú presnosť. Okrem toho sme testovali výkon algoritmu pri zvyšúcom sa počte nových tried. Nakoniec sme použili tento algoritmus na tréning a vyhodnotenie modelu pomocou snímkov z kamery lavého oka robota NICO.

Literatúra

- [1] Xin Wang, Thomas E Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. Frustratingly simple few-shot object detection. *arXiv preprint arXiv:2003.06957*, 2020.
- [2] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.
- [3] Roboflow. Roboflow: The world’s easiest computer vision platform. <https://roboflow.com/>.
- [4] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.
- [5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*, volume 1, pages 886–893. Ieee, 2005.
- [6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

- [7] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.
- [8] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.
- [9] Léon Bottou. On the computational efficiency of training neural networks. In *Proceedings of the 14th International Conference on Neural Information Processing Systems*, pages 146–152, 2001.
- [10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. In *Nature*, volume 323, pages 533–536. Nature Publishing Group, 1986.
- [12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.
- [14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.
- [16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9. IEEE, 2015.
- [17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.
- [18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [19] TzuTa Lin. labelimg. <https://github.com/tzutalin/labelImg>, accessed April 2023.
- [20] Viktor Kocur. planar_nico_vision. https://github.com/kocurvikk/planar_nico_vision, accessed April 2023.
- [21] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. Ieee, 2001.
- [22] Guangxing Han, Shiyuan Huang, Jiawei Ma, Yicheng He, and Shih-Fu Chang. Meta faster r-cnn: Towards accurate few-shot object detection

with attentive feature alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 780–789, 2022.

Zoznam obrázkov

1.1	Konkrétny príklad aplikácie filtra na vstup	8
1.2	Znázornenie paddingu pri konvolúcii	9
1.3	Aplikácia dropoutu	14
1.4	Architektúra Faster R-CNN	16
1.5	Model pre frustratingly simple few shot object detector . . .	23
2.1	Príklady anotovaných obrázkov pre triedu apple	26
2.2	AP50 na testovacích dátach pre jednotlivé triedy pre 1-shot detekciu.	28
2.3	Graf presnosti na testovacích dátach pre 1-shot detekciu. . . .	29
2.4	AP50 na testovacích dátach pre jednotlivé triedy pre 2-shot detekciu.	30
2.5	Graf presnosti na testovacích dátach pre 2-shot detekciu. . . .	30
2.6	AP50 na testovacích dátach pre jednotlivé triedy pre 3-shot detekciu.	31
2.7	Graf presnosti na testovacích dátach pre 3-shot detekciu. . . .	32
2.8	AP50 na testovacích dátach pre jednotlivé triedy pre 5-shot detekciu.	33
2.9	Graf presnosti na testovacích dátach pre 5-shot detekciu. . . .	33

2.10 AP50 na testovacích dátach pre jednotlivé triedy pre 10-shot detekciu.	34
2.11 Graf presnosti na testovacích dátach pre 10-shot detekciu.	35
2.12 AP50 na testovacích dátach pre jednotlivé triedy pre 15-shot detekciu.	36
2.13 Graf presnosti na testovacích dátach pre 15-shot detekciu.	36
2.14 Novel AP50 na testovacích dátach vzhladom k počtu trénovacích obrázkov.	37
2.15 Čas tréningu vzhladom k počtu trénovacích obrázkov.	38
2.16 Najnižia AP50 na testovacích dátach vzhladom k počtu trénovacích obrázkov.	39
3.1 Celková strata počas 1-shot fine tuningu	41
3.2 Porovnanie mAP50 na testovacích dátach 5 min. tréningu s plným tréningom.	42
3.3 Porovnanie mAP50 na testovacích dátach 20 min. tréningu s plným tréningom	43
3.4 Porovnanie presnosti pôvodného a zrýchleného algoritmu pomocou zapamätania výstupu z Backbone a prispôsobenia obrázkov na rovnakú veľkosť	46
3.5 Porovnanie presnosti pri batch size = 1 a batch size = 2'.	47
3.6 Graf porovnania mAP50 na testovacích dátach pôvodného algoritmu a 1. pokusu o zrýchlenie pri batch size = 1.	49
3.7 Graf porovnania mAP50 na testovacích dátach pôvodného algoritmu a 2. pokusu o zrýchlenie pri batch size = 1.	50
3.8 Graf porovnania mAP50 na testovacích dátach pôvodného a zrýchleného algoritmu.	51

3.9	Porovnanie presnosti na testovacích dátach a dĺžky tréningu rôznych batch size.	53
3.10	Porovnanie rýchlosťí pôvodného algoritmu a algoritmu po zrýchlení	54
4.1	Príklady anotovaných obrázkov pre triedu zebra	57
4.2	Príklady anotovaných obrázkov pre triedu pomaranč	57
4.3	Príklady anotovaných obrázkov pre triedu nákladné vozidlo . .	58
4.4	Príklady anotovaných obrázkov pre triedu medveď	58
4.5	Graf presností na testovacích dátach pre 10 novel tried + 15 base tried pri 10-shot detekcii.	59
4.6	AP50 na testovacích dátach pre 10 novel tried pri 10-shot detekcii	59
4.7	Príklady anotovaných obrázkov pre triedu žirafa	60
4.8	Príklady anotovaných obrázkov pre triedu dážnik	61
4.9	Príklady anotovaných obrázkov pre triedu futbalová lopta . .	61
4.10	Príklady anotovaných obrázkov pre triedu basebalová pálka .	62
4.11	Príklady anotovaných obrázkov pre triedu vydlička	62
4.12	Graf presností na testovacích dátach pre 15 novel tried + 15 base tried pri 10-shot detekcii.	63
4.13	AP50 na testovacích dátach pre 15 novel tried pri 10-shot detekcii.	63
4.14	Príklady anotovaných obrázkov pre triedu nôž	64
4.15	Príklady anotovaných obrázkov pre triedu banán	65
4.16	Príklady anotovaných obrázkov pre triedu mrkva	65
4.17	Príklady anotovaných obrázkov pre triedu hotdog	66
4.18	Príklady anotovaných obrázkov pre triedu počítačová myš .	66
4.19	Graf presností na testovacích dátach pre 20 novel tried + 15 base tried pri 10-shot detekcii.	67

4.20 AP50 na testovacích dátach pre 20 novel tried pri 10-shot detekcii.	68
4.21 Porovnanie presnoti na testovacích dátach 10-shot detekcie s 20 novel triedami + 15 base triedami s pôvodným a dvojnásobným počtom iterácií	69
4.22 Presnosť 10-shot detekcie na testovacích dátach vzhľadom k počtu novel tried	70
4.23 Graf porovania presnosti na testovacích dátach pre fine-tuning bez a s base triedami pri 10-shot detekcii	71
5.1 Príklad trénovacích obrázkov z kamery robota NICO.	74
5.2 Príklad úspešných detekcií testovacích obrázkov z kamery robota NICO.	75
5.3 Príklad neúspešných detekcií testovacích obrázkov z kamery robota NICO.	76
5.4 Príklad trénovacích obrázkov z kamery robota NICO zbavených skreslenia.	79
5.5 Príklad úspešných detekcií testovacích obrázkov z kamery robota NICO zbavených skreslenia.	80
5.6 Príklad neúspešných detekcií testovacích obrázkov z kamery robota NICO zbavených skreslenia.	81